WHITE PAPER
ON
ULTRA-DEPENDABLE ARCHITECTURES

Dr. Daniel P. Siewiorek

Chairman

Carnegie Mellon University

Dr. M. Y. Hsiao

International Business Machines

Prof. David Rennels

University of California Los Angeles

Dr. James Gray

Tandem Computer, Inc.

Dr. Thomas Williams

International Business Machines

August 15, 1988

Revised September 30, 1988

Final Revision - November 2, 1988

EXECUTIVE SUMMARY

Since there are many mission-critical DOD requirements for ultra-dependable systems, most of which must, by their nature, be distributed, it is essential to establish a development process that makes routine the design/construction of systems with an almost arbitrarily specified dependability[1] and supports analysis of the cost/benefit trade-offs for obtaining this level of dependability.

The current state of the art in dependable system design can be summarized as:

- While hardware design principles, algorithms and techniques for design to near arbitrary cost and dependability specifications exist for isolated systems, they are either not implemented or not widely applied.

- The management of information in arbitrarily dependable distributed systems is not well established, even when the designer has control over communication links.

- For geographical dispersed systems in which the designer has no control over communication links there currently does not exist the capability for designing and developing systems to arbitrary dependability specifications.

The ISAT recommends the following research thrusts:

1. Development and demonstration of a design methodology and environment for ultra-dependable, time-critical isolated systems including software and security.

[1] Dependability is a general term encompassing a host of "-abilities" including: reliability, availability, maintainability, diagnosability, etc.

2. A project to develop an ultra-dependable, time-critical distributed system with development of supporting research.

I. SCOPE

The aim of this research is to develop the methodology needed to implement high performance computer systems with very high dependability. Dependability is a general term encompassing reliability, availability, maintainability, diagnosability, etc. The ultimate goal is to produce a system which neither crashes nor fails during a ten-year operational life. Furthermore, these systems should require less than five percent additional resources than contemporarily designed systems.[2] The term "system" is used to denote a computer and the entities it controls. The system could be a single isolated computer complex or a distributed system.

While spacecraft computers have been built that demonstrate survivability i.e., for ten years, these computers are in the range of 100 kilo operations per second (KOPS) and require substantial ground intervention to "work around" failures. Due to simplicity in hardware and software, space-borne computers exhibit Mean Time Between Crashes (MTBC) measured in one to two years. Conventional ground-based commercial systems have MTBC's in the range of 100 to 1000 hours. While contemporary space-borne computer systems exhibit adequate reliability, this reliability is achieved with substantial ground-based human intervention and up to three orders of magnitude slower processing performance. While commercial computers have adequate processing power, they do so at up to three orders of magnitude shorter MTBC and/or unavailability. Commercial systems dedicated to a single application (e.g. telephone switching, transaction processing) are closing the availability gap between space-borne and commercial systems.[3] A meaningful metric is the Mean Number of Instructions to Restart (MNIR) which is a function of the processing rate and rate of system restarts after an outage. Contemporary space-borne especially designed commercial systems are on the order of 10^{13} to 10^{14} MNIR. The goal

[2] Does not apply to long life unmaintained systems.

[3] Manufactures of fault tolerant computers, such as Tandem and Stratus, have achieved MTBC's measured in years.

should be four orders of magnitude improvement i.e., 10^{17} to 10^{18} MNIR in the next 10 years.[4]

A computing system progresses through many stages during it's life. These stages include specification, logic design, prototype debugging, manufacturing, installation, and field operation. Deviations from intended behavior, or errors, 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 the system's operational life, 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 the humidity of the computer earth ground loops. Inconsistent states can also be caused by operator errors and by design errors in hardware or software. The vast majority of prior research has been focused on physical hardware failure during the operational life of the system. As the scale of integration of hardware has increased, the corresponding failure rate per unit function has decreased. Transient and intermittent errors occur 10 to 100 times more frequently than permanent failures. Furthermore, measurements from commercial high-reliability systems indicate that hardware-related problems contribute less than 50% of the reliability-related problems.

Thus, research needs to focus equally on all stages of the system's life as well as all aspects of a system (e.g., hardware, software, environment, human operators, etc.). In particular, design errors - whether they be in hardware and/or software - become a particularly elusive problem. Closely tied to the human creative process, design errors are difficult to predict. Gathering statistical information about the phenomena is difficult

[4] As performance increases MNIR will have to be improved just to maintain constant availability.

because each design error occurs only once per system. The rapid rate of development in hardware technology constantly changes the set of design trade-offs, further complicating the study of hardware design errors. Once these systems have been designed and built, one is left with the challenge of verifying that the design goal has been met. For example, a recent aerospace project had a goal of probability of error of 10^{-10} per hour. This goal translates into more than a million years of error-free operation. One cannot observe a thousand systems for a thousand years to decide whether such a goal has been met. A pro-active validation methodology is required.

In order to achieve the goals outlined above, a number of "break throughs" will be required. A partial list of potential break throughs might include:[5]

- Development of techniques and practices to minimize design errors. Fault-avoidance techniques such as defensive programming and design verification techniques such as mutation testing have already been applied to hardware and software design. However these techniques are only marginally effective in detecting rarely-occurring interaction of three or four anomalous conditions that typify a design error. Formal proofs of correctness of cleanly decomposed formal specifications is another approach. Furthermore, an infra structure of CAD and CASE tools to be developed to support design of ultra-dependable architectures (hardware and software).

- New hardware architectures, software architectures, and algorithms for fault-detection, diagnosis, recovery, and reconfiguration in distributed systems are required. The current techniques rely on brute-force replication that is costly in either physical or design resources. Stronger links need to be forged with other fields of computer technology including: programming methodology, language

[5] Note that research is under way in many of these areas.

design, compilers, database, artifical intelligence, and security. A design methodology must be developed which treats multiple objectives (i.e., performance, programmability, dependability, and security) uniformly. Design for dependability should extend seamlessly through hardware, programming, system design, and operator interaction.

- Improved techniques for concurrent error-detection are required. Data from commercial systems indicate that the single biggest source of downtime is undetected or misdiagnosed errors.[6] The use of techniques to monitor data from an operating system and predict failures prior to catastrophic results would allow the system to adapt prior to catastrophic failures.[7] Techniques are required to detect non-classical faults (e.g., changes in timing and state behavior of component elements, multiple hard/soft/design errors, etc.) which occur more frequently than usually assumed.

- Systems must be designed: for real-time recovery and repair; and to identify and preserve critical state under various fault conditions.

- As computing systems become more complicated to operate, mechanisms are required to minimize or prevent operator mistakes. In a remote terminal environment, an operator may be an ordinary system user.

[6] Errors from all sources including hardware, software, environment, operation, maintenance, etc.

[7] A recent study of SUN file servers on the CMU network demonstrated that when there are three error log entries relating to a physical failure, there is greater than a 90% probability of predicting a failure an average 200 hours prior to catastrophic failure. However, one-third of the physical failures and no error log entries at all indicating either catastrophic failure or, in our case, inadequate error detection mechanisms.

- Systems must be designed/built to be immune to environmental noise and interferences.

II. BACKGROUND

There is a close tie between architecture and high dependability. Let us examine some of the attributes of computer and network architecture. Data processing elements can range from tightly-coupled, high-cooperative to self-autonomous. Many combat situations are based upon autonomous units operating on self-initiative. Prior to a battle, a foot soldier or fighter pilot will be briefed and drilled on the plan for battle. Communications during the phase can be very intense with exchange of information, opinions, and intuition. During the actual combat, however, communications are undesirable whether due to security and detection, or due to the chaos of the battlefield. Decisions are made locally based upon knowledge of the battle plan and limited inputs from locally-available sensors (in the case of a soldier, his eyes and ears). Not much research has been performed into systems which change modes between high and low available communications bandwidth. Thus research should be done in both types of systems as well as a hybrid system which switches between both modes. While the communications between nodes in the network may be highly variable, we can assume high performance communications internal to a network node. These network nodes should also be a hybrid of various forms of computations.

Table 1 illustrates four types of computations as a function of the grain size - the period between synchronization events for multiprocessors or processing elements. Synchronization is necessary in parallel processing to initialize the task, parcel out work, and merge results.

Table 1 - Simplified taxonomy of parallel architectures as a function of inter-computation communication interval

Grain Size	**Construction for Parallelism**	**Synchronization Interval (Instructions)**
Fine	Parallelism inherent in single instruction or data stream	1
Medium	Parallel processing or multi-tasking within a single process	100 - 1000
Coarse	Multiprocessing of concurrent processes in a multiprocessing environment	1000 - 10,000
Very Coarse	Distributed processing across network nodes to form single computing environment	10,000 - 1 million

The very coarse grain size corresponds to a network. The individual computing nodes should be able to support fine, medium, and coarse grain computations. Medium and coarse grain computations can be handled by multiprocessors. Fine grain computation requires specialized processors such as systolic or array processors. A unified architecture supporting fine, medium, and coarse grain computations would be versatile enough to handle the widest range of anticipated applications.

There is little or no experience with such a unified architecture. Existing systems handle one, or at most two, grain sizes. Most experience has been with very coarse or fine grain size. Multicomputers such as Tandem and Stratus are tuned to very coarse grain size. Dedicated signal or array processors handle fine grain size. Commercial multiprocessors should process medium and coarse grain sizes.

Over the past two decades there has been a steady growth in the number of fault-tolerant systems. The Bell System's Electronic Switching System (ESS) is a pioneer in this area. Introduced in 1965, ESS-1 required over ten years of field experience to reach it's goal of three minutes downtime per year. A study of causes of downtime attributed 20% to hardware, 15% to software, 35% to insufficient error detection or incorrect error diagnosis, and 30% to operational (e.g. human) mistakes. In 1976 Tandem introduced a highly available commercial transaction processing system. In a study conducted by Jim Gray from 1985 to 1987 over 80% of the Tandem customers experienced no unscheduled outages during those two-years. Mean time to system outage was in excess of four years. Table 2 summarizes experiences in the ESS, Tandem, and some commercial systems.

Table 2 - Sources of Outage

	ESS* [Toy '87]	Japanese [Users] [1985]	Tandem [Gray '85]	Tandem [Gray '87]	Commercial Lg./Med. Systems [Users] [1988]
Hardware	.2	**	.18	.19	.45
Software	.15	**	.26	.43	.20
Maintenance	-	**	.25	.13	.05
Operations	.65	.11	.17	.13	.15
Procedural Errors	.3				
Recovery Deficiencies	.35				
Environment	-	.13	.14	.12	.15

*Fraction of downtime

** 0.75 for all three areas including all vendor-supplied items including hardware, software, and maintenance. The report did not break the vendor source of outage into its constituent parts.

As we can see, the source of outage is distributed among the areas identified in the missing potential "break throughs" list in Section 1. These successes have come in dedicated application areas (e.g. telephone switching and transaction processing) in which there are many aspects of the application which contribute to a clean fault-tolerant architecture. The Pluribus IMP for the DARPAnet is a good example of a dedicated system. A store and forward node in a switching network only requires good error-detection mechanisms. The rest of the network buffers the information and if a Pluribus should fail, it need only reconfigure itself into a working system and the rest of the network will retry their accesses. Thus to a first order approximation the Pluribus IMP need not worry about data loss during reconfiguration since this information will be supplied by the rest of the distributed system. More general purpose techniques are required - techniques that are not so dependent upon the characteristics of the application. However, a system that utilizes the characteristics of an application has a higher dependability potential at less cost than a general purpose system which knows nothing about the structure of its application. Therefore, the ultra-dependable architecture/system can take the advantages of applicaton dependencies, e.g., an image processor system may not require the same degree of dependability as a commercial control system.

A number of research groups have targeted reliability as a major thrust. A partial list includes

- Center for Dependable Systems (CDS). A group of researchers at CMU looking at hardware/software theory/practice of dependable system design. Data collection, modelling, and validation techniques have been developed. Dan Siewiorek is Director.

- Center for Reliable Computing (CRC). A group of researchers at Stanford University focusing on testing and on-line error detection. Statistical studies of SLAC facilities have been performed. Ed McCluskey is Director.

- UCLA/JPL. Looking back to the early 1960's, the JPL/UCLA cooperation has had a direct impact upon the architecture of unmanned deep space probes. It's ultra-reliable systems have severe volume, power, and weight restrictions. Dave Rennels and Al Avizienis are the primary contacts at UCLA.

- University of Newcastle upon Tyne. Brian Randall has a group working on software approaches to high dependability. In partricular, attention has focused upon minimizing software design errors.

- University of Texas. Jacob Abraham and Mirek Malek have explored issues in algorithm-specific error detection, testing, and parallel architectures.

There are many other organizations that either have one or two researchers or had a large group that is no longer active that have contributed to the theory of high dependability systems. Some of these organizations include SRI (Jack Goldberg is still very active), Johns Hopkins (Gerry Masson), University of Michigan (John Hayes and John Myers), University of Illinois (Ravi Iyer), Draper Labs (Jay Lala), IBM (M.Y. Hsiao, T. Basil Smith), NASA Langley's AirLab facility, Ballistic Missile Defense Architectural Research Center, etc. There is growing interest in high dependability abroad (i.e., Japan, France, and Germany).

III RESEARCH OPPORTUNITIES

A substantial amount of research and development is required in the area of fault-tolerant architectures. The techniques built into contemporary systems have their origin in the 1960's. These techniques are based upon the brute force of replication. Duplication has been used in cold (e.g. unpowered spares in satellites), warm (e.g. checkpointing or shadowing in Tandem), and hot (e.g. simultaneous processing ESS, Stratus) standby configurations. Duplication is only as effective as the ability of added hardware and/or software to correctly detect and diagnose the faulty unit. Triplication and voting removes the necessity to diagnose and has been used in both hardware (e.g. August Systems) and software (e.g. Space Shuttle computers). Reconsider the sources of unavailability found in the ESS studies. Replication only addresses the 20 percent of unavailability due to hardware and a portion of the 35 percent due to detection and diagnosis inefficiencies.[8] Not only are new architectural concepts required, but also research into reducing: errors in software, inadequate error detection/diagnosis, and operational mistakes. Consequently research is required into the methodology of design. Here we mean design in its most general terms, encompassing hardware, operating system, application software, and operating procedures. Reliability and fault tolerance needs to be designed in from the start. This is why new commercial vendors (e.g. Tandem) can be successful while existing vendors have a difficult time retro-fitting these features. Cost-effective solutions often depend on what is already existing in the design. Furthermore, applications often have characteristics which can be used to advantage. For example, real-time systems (such as telephone switching) do not need to store sensor data for long periods of time. The real-time data quickly re-initializes the data base. The system merely needs to establish a working configuration. This is in contrast to a transactions-processing system where the data base has to be correct, even through system crashes.

During the system design process there is a large gap between designers and modelers. The modeler develops mathematical models which require abilities beyond those possessed by designers (such as "fail fast" processors). Furthermore, modelers use abstract parameters (such as coverage - the probability that given an error, the system correctly detects, diagnoses, and recovers from it) that are difficult or impossible to derive from designs. Thus research is required to develop sophisticated Computer-Aided Design (CAD) tools for dependability analysis. These tools are to read data from the design data base and produce dependability (i.e., reliability, availability, maintainability, diagnosability, etc.) estimates upon requests. Due to the partial and inconsistent state of the

[8] It should be noted that a NASA fault-injection study indicated that a duplex computer detected only 60 percent of the injected faults since the applications program did not exercise all of the hardware. A common assumption is that duplication detects 100 percent of failures.

evolving design data base, these CAD tools must be able to make educated estimates required for the mathematical models. Furthermore, these tools should be capable of detailed analysis or simulation to derive complex metrics such as coverage. There tools exist only in fledgling form today with no tools integrated into a design data base.

Since characteristics of the application will be used for enhancing fault tolerance, a fault simulator to handle actual application code in different redundancy architectures needs to be developed. Consider duplication. The application code can use watchdog timers to detect errors or comparison to an active copy. The simulator should answer questions such as the effectiveness, in a mathematical sense, of each approach to fault detection as well as the manifestation of undetected faults. The manifestations of undetected faults could be used to drive a higher level simulator which predicts the effectiveness of the whole system.

Algorithms and heuristics for control in distributed systems need to be developed. Not only will delays in communication be uncertain, but data may be lost or even contradictory. Since the system must operate in real time, effective means of detecting overload and shedding load must be developed. Contemporary operating systems either do not deal with these problems or only deal with them in an extremely limited way. A substantial amount of basic research is required in this area. A few hand-crafted ad hoc fault tolerant distributed systems have been developed. However there is no cohesive theory on how to design systems with distributed state (e.g., information) that may be too time consuming to collect into one place or may not be accessible due to partitioning.

Demonstration of system-effectiveness will be a major concern. Unclassified algorithms which represent typical work loads should be made available to researchers. If the exact algorithms cannot be made available, a representation of the computational resources and their interdependencies (such as represented by a data flow graph) could serve as a synthetic work load.

A testing strategy needs to be developed. Strategies should be based upon observation of the system during normal processing. The historic approach of running diagnostics is inadequate for uncovering the unexpected. In addition to normal system workload some random stress should be added to the system to accelerate the discovery of unexpected phenomena. Research to support the testing activity includes on-line error detectors both in hardware and software, random stimulus generation, logging of detected anomalies, and an automatic on-line analysis of the system at its peak processing rate since systems are more prone to error when heavily stressed.

In order to demonstrate the effectiveness of the above verifications and validation (V&V) process, a testbed should be developed. The effectiveness of the V&V methodology could be demonstrated by applying it to existing commercial fault-tolerant systems.

Substantial work needs to be done in the areas of security, fault tolerance, and parallel processing. Security and fault tolerance are mutually supportive concepts. It is not possible to have a secure system that is not reliable, while on the other hand the integrity checking of security is yet another means of error detection which can be used to improve fault tolerance. There is very little practical experience programming parallel processors for a large, single, homogeneous application. The extra dimension of parallelism increases the difficulty of validating the system. In addition, error detection and recovery mechanisms represent an added complexity that must also be validated.[9]

[9] Redundancy management code is frequently more complex than the application code. Furthermore, it is substantially more difficult to debug than real time interrupt-driven code since an error can occur any physical place at any time.

Finally, there should be a technology assessment program which provides information on expected capabilities of technology to the system designers. This includes sensor, communications, and data processing technologies. In particular, non-silicon technologies such as gallium arsenide, electro-optics, and magnetic materials should be explored for their unique capabilities such as resistance to radiation, high bandwidth, and/or high densities.

IV. IMPACT

The culmination of this research could produce systems which never fail during their life expectancy. Dependency on computing system has grown so great that it is becoming difficult or impossible to return to less sophisticated mechanisms. When an airline seat selection computer "crashes", the airline can no longer revert to assigning seats from a manual checklist; since the addition of round trip check-in service, there is no way of telling which seats have been assigned to passengers who have not yet checked in without consulting the computer. More and more of the ways that the United States conducts business have become intimately intertwined with computer technology. Note that we have become dependent upon electrical grids and telephone information networks that are constantly available. We need to develop a national-wide information utility which is constantly available. Todays computer networks have taken fledgling steps in this direction. However, as we grow more interconnected, we must place safeguards to prevent the equivalent of a "northeast blackout". Even innocent actions such the broadcast of cartoons through chained mailing distribution lists can bring a sophisticated network to it's knees. Malicious actions such as computer viruses can have potentially damaging consequences. However, a virus can be viewed as a generalized "error" from which mechanisms in a high reliability system would provide safeguards.

Dependability is synergistic with several other aspects of computer science including inconnectivity, security, productivity, and ease of use. There are several advantages of a high dependability program to DoD:

- Increase the availability of weapons systems which is presently unacceptably low.
- Construction of systems with very long unattended lifetime (e.g., inaccessible systems).
- Construction of systems which survive extreme levels of damage or severe security attacks.

Eventually, all computers will be highly dependable. However, industry will not likely develop techniques appropriate to DoD requirements. An initiative in high dependability systems could make DARPA a leader in meeting the challenges of the new found awareness in systems that never fail. Currently most DoD systems managers focus on performance/functionality rather than sacrifice resources on what are perceived as rare events (with possibly horrible consequences).

V. CONCLUSION AND RECOMMENDATION

It is suggested that a program be developed composed of three phases. The first phase of approximately two years duration should consist of small contracts to individual universities and companies to provide the basic technology in areas such as parallel processing, security, and testing. The second phase of approximately three years duration should include teams of contractors to develop system concepts in simulators to verify those concepts. Below is a representative list of research projects for the first two phases.

Phase 1 - Basic Technologies

Architecture

- New architectures based on multiple modes of available communications bandwidth.
- New fault-tolerant techniques.
- Interaction of security and reliability.
- Distributed management in the face of errors.
- Design and evaluation of effectiveness of error detectors and error diagnosis.
- Design for recoverability
- Computer-Aided Design (CAD) tools for evaluating dependability and synthesizing fault tolerant tecnhiques integrated with more traditional CAD and CASE (Computer Aided Software Engineering) tools.
- Fault Tolerant software techniques.
- Expert systems to "operate" and diagnose systems.

Applications

- Programming experience in real-time, distributed, hybrid systems. Especially multiprocessors.
- Techniques to minimize operational (procedural) mistakes.
- Heuristics for autonomous operation.

Verifications and Validation

- Develop simulation techniques for generating rarely occurring sequences so that design functionality and error recovery can be verified.
- Demonstrate methodology on an existing architecture.

Phase 2 - System's Concepts

Architecture

- Develop a simulator for entire system.
- Measure performance.
- Inject faults into simulator to measure effectiveness of architecture, measure performance in the presence of errors.
- Use a commercial fault-tolerant architecture for fault injection studies.

Verification and Validation

- Develop a simulator for combinations of application code and fault-tolerant architectures. Develop a method for evaluating these combinations. The results would be input to system models.

- Measure with and without faults the synthetic workload/operating system/commercial architecture produced above.

The final phase is a mission-oriented demonstration that should last five years and be composed of two subphases. The first subphase of two years duration is a system architecture development supported by a detailed simulation upon which application software can be run. The second subphase includes actual construction with off-the-shelf chips, boards, and software required to assemble a complete prototype. The prototype should be heavily instrumented both in hardware and software to not only facilitate system debugging, but also to aid in system tuning and identification of system bottlenecks. There should be at least two system contract teams in this phase with each team composed of several industrial and university members. A subset of each contractor group should be devoted to devising test cases not only for their own architecture, but also for the other contract group's architecture. These competing "tiger teams" will provide an independent test of each prototype's capability. "Tiger teams" have been very effective in the past in probing the capabilities of secure systems. It may even be desirable to have small contracts with independent groups that are not part of the prototype development to serve as "tiger teams" developing tests for the prototypes prior to prototype acceptance.