International Journal of Current Research and Modern Education (IJCRME)
ISSN (Online): 2455 - 5428
(www.rdmodernresearch.com) Volume I, Issue II, 2016

PERFOMANCE ANALYSIS AND FAULT TOLERENCE OF SOFTWARE ENVIRONMENT

S. Umamageswri* & T. Arulselvam**

* Assistant Professor, Department of Computer Science, Thiru Kolanjiappar Government Arts College, Vridhachalam, Tamilnadu

** Assistant Professor, Department of Computer Science, Thiru Kolanjiappar Government Arts College, Vridhachalam, Tamilnadu

Abstract:

Fault tolerance is the characteristic of a system that tolerates the class of failures. It will analysis the performance and enables a system to continue its intended operation, possibly at a reduced level, rather than failing completely, when some part of the system fails. The system as a whole is not stopped due to problems either in the hardware or the software. [1] [2] Fault tolerance is the ability for software to detect and recover from a fault that is happening or has already happened in either the software or hardware in the system in which the software is running in order to provide service in accordance with the specification. [3] Fault tolerance is not a solution unto itself however, and it is important to realize that software fault tolerance is just one piece necessary to create the next generation of systems. A highly fault-tolerance system might continue at the same level of performance even though one or more components have failed. For example, a building with a backup electrical generator will provide the same voltage to wall outlets even if the grid power fails [4].

Key Words: Fault Tolerance of Software, Redundancy, Design & Response Stages of Faults.

1. Introduction:

The aim of fault-tolerance design is to minimize the probability of failures, whether those failures simply annoy the customers or result in lost fortunes, human injury or environmental disaster. [5] [6] Fault tolerance is the ability of a system to continue performing its intended function in spite of faults. In a broad sense, fault tolerance is associated wither liability, with successful operation, and with the absence of breakdowns. A fault-tolerant system should be able to handle faults in individual hardware or software components, power failures or other kinds of unexpected disasters and still meet its specification. [7] Fault tolerance is needed because it is practically impossible to build a perfect system.

The system is designed and implemented perfectly; faults are likely to be caused by situations out of the control of the designers. Failure can be a total termination of function, or a performance of some function in as abnormal quality or quantity, like deterioration or instability of operation.

1.1 Basic Concepts:

The role of fault tolerance in distributed systems we first need to take a closer look at what it actually means for a distributed system to tolerate faults. Being fault tolerant is strongly related to what are called dependable systems. Dependability is a term that covers a number of useful requirements for distributed systems including the following.

- ✓ Availability
- ✓ Reliability
- ✓ Safety
- ✓ Maintainability

Availability [8] [9] is defined as the property that a system is ready to be used immediately. In general, it refers to the probability that the system is operating correctly at any given moment and is available to perform its functions on behalf of its users. In other words, a highly available system is one that will most likely be working at a given instant in time.

Reliability [8] [9] refers to the property that a system can run continuously without failure. In contrast to availability, reliability is defined in terms of a time interval instead of an instant in time. A highly reliable system is one that will most likely continue to work without interruption during a relatively long period of time. This is a slight but important difference when compared to availability. If a system goes down on average for one, seemingly random millisecond every hour, it has an availability of more than 99.9999 percent, but is still unreliable. Similarly, a system that never crashes but is shut down for two specific weeks every August has high reliability but only 96 percent availability. The two are not the same.

Safety [8] [9] refers to the situation that when a system temporarily fails to operate correctly, no catastrophic event happens. For example, many process-control systems, such as those used for controlling nuclear power plants or sending people into space, are required to provide a high degree of safety. If such control systems temporarily fail for only a very brief moment, the effects could be disastrous. Many examples from the past (and probably many more yet to come) show how hard it is to build safe systems.

Finally, **maintainability** [8] [9] refers to how easily a failed system can be repaired. A highly maintainable system may also show a high degree of availability, especially if failures can be detected and repaired automatically.

2. Origins of Faults:

Failures are caused by errors and errors are caused by faults. Faults are, in turn, caused by numerous problems occurring at specification, implementation, and fabrication stages of the design process [10] [11].

We can classify the sources of faults into four groups:

- ✓ incorrect specification
- ✓ incorrect implementation
- ✓ fabrication defects
- ✓ External factors.

Incorrect specification results from incorrect algorithms, architectures, or requirements. Faults caused by incorrect specifications are usually called specification faults.

Faults due to incorrect implementation, usually referred to as design faults, occur when the system implementation does not adequately implement the specification.

A source of faults in hardware is component defects [12]. These include manufacturing imperfections, random device defects and components wear-outs. Fabrication defects were the primary reason for applying fault-tolerance techniques to early computing systems, due to the low reliability of components.

Fourth causes of faults are external factors, which arise from outside the system boundary, the environment, the user or the operator.

3. Classification of Fault:

Generally Faults can be classified into one of three categories [13] [14]:

Transient Faults: these occur once and then disappear. For example, a network message transmission times out but works fine when attempted a second time.

Intermittent Faults: these are the most annoying of component faults. This fault is characterized by a fault occurring, then vanishing again, then occurring An example of this kind of fault is a loose connection.

Permanent Faults: this fault is persistent: it continues to exist until the faulty component is repaired or replaced. Examples of this fault are disk head crashes, software bugs, and burnt-out hardware.

4. Fault Tolerance and Redundancy:

There are various approaches to achieve fault-tolerance. Common to all these approaches is a certain amount of redundancy. For our purposes, [15] *redundancy* is the provision of functional capabilities that would be unnecessary in a fault free environment.

Two kinds of redundancy are possible:

- ✓ Time Redundancy
- ✓ Space Redundancy

Time Redundancy the timing of the system is such, that if certain tasks have to be rerun and recovery operations have to be performed, system requirements are still fulfilled.

Space Redundancy provides additional components, functions, or data items that are unnecessary for a fault-free operation. Space redundancy is further classified into Hardware, Software and Information Redundancy depending on the type of redundant resources added to the system.

Hardware Redundancy: [16] [17] the system is provided with far more hardware than needed for basic functionality. Hardware redundancy is the use of additional hardware to compensate for failures:

- ✓ Fault detection, correction, and masking, multiple hardware units are assigned to the same task in parallel and their results compared.
  - o Detection: if one or more (but not all) units are faulty, this shows up as a disagreement in the results (even byzantine faults can be detected).
  - o Correction and masking: if only a minority of the units is faulty, and a majority of the units produce the same output, the majority result can be used to correct and mask the failure.
- ✓ Replacement of malfunctioning units: correction and masking are short-term measures. In order to restore the initial performance and degree of fault-tolerance, the faulty unit has to be replaced.

Hardware redundancy is a fundamental technique to provide fault-tolerance in safety-critical distributed systems: aerospace applications, automotive applications, medical equipment, some parts of telecommunications equipment, nuclear centers, military equipment, etc.

Software Redundancy: [18] the system is provided with different software versions:

- ✓ Results produced by different versions are compared;
- ✓ When one version fails another one can take over.

There are several aspects which make software very different from hardware in the context of redundancy: A software fault is always caused by a mistake in specification or by a bug. (a design error).

- ✓ No software faults are produced by manufacturing, aging, stress, or environment.
- ✓ Different copies of identical software always produce the same behavior for identical inputs

Information Redundancy: data are coded in such a way that a certain number of bit errors can be detected and, possibly, corrected (parity coding, checksum codes, cyclic codes).

5. Techniques for Fault Tolerance in Software:

Faults in software are divided into three main categories [6]:

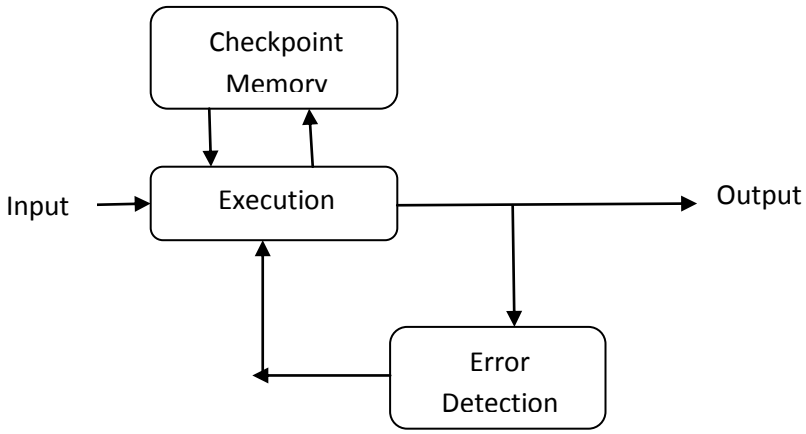
Fault Avoidance/Prevention: This include design methodologies which attempt to make software provably fault-free. Fault avoidance or prevention techniques are dependability enhancing techniques [11] employed during software development to reduce the number of faults introduced during construction. These avoidance, or prevention, techniques may address, for example, system requirements and specifications, software design methods, reusability, or formal methods. Fault avoidance techniques contribute to system dependability through rigorous specification of system requirements, use of structured design and programming methods, use of formal methods with mathematically tractable languages and tools, and software reusability.

Fault Removal: These methods aim to remove faults after the development stage is completed. This is done by exhaustive and rigorous testing of the final product.

Fault Tolerance: This method makes the assumption that the system has unavoidable and undetectable faults and aims to make provisions for the system to operate correctly even in the presence of faults.

- ✓ **Software fault tolerance is basically divided into two groups**: [19] [20] single version and multi version software techniques. Single version techniques are concerned with single software by adding several types of mechanisms during the design phase, with a goal to detect, contain, and handle errors. Multi-version fault tolerance techniques use multiple versions of the same software in a structured way to ensure that design faults in one version do not cause system failure.
- ✓ **Single Version Software Fault Tolerance Techniques:** Single-version fault tolerance is based on the use of redundancy applied to a single version of a piece of software to detect and recover from faults. Among others, single-version software fault tolerance techniques include considerations of program structure and actions, error detection, exception handling, checkpoint and restart, process pairs, and data diversity. For [7] single-version software, there are few recovery mechanisms. The most useful mechanism is the checkpoint and restart mechanism. A restart or backward error recovery has the advantage of being independent of the damage caused by a fault, applicable to unanticipated faults, general enough to be used at multiple levels in a system, and conceptually simple. There are two types of restart recovery: static and dynamic. A static restart recovery is based on returning the module of software to a predetermined state. This can be a direct return to the initial reset state, or to one of a set of possible states. The selection is based on the operational situation at the moment the error detection occurred. Dynamic restart uses dynamically created checkpoints that are snapshots of the state at various points during the processing. Checkpoints can be created at fixed intervals or at particular points during the computation, determined by an optimization rule.
- ✓ **Multi-Version Software Fault Tolerance Techniques:** [22] Design diverse techniques are used in a multiple variant software environment and utilize functionally equivalent yet independently developed software versions to provide tolerance to software design faults. Examples of such techniques include

recovery blocks, N-version programming. There are two strategies for software fault tolerance.



- ✓ **Error Processing**: error removal, before failure occurs

Error detection: identification of erroneous state(s)
Error diagnosis: damage assessment
Error recovery: error-free state substituted to erroneous state

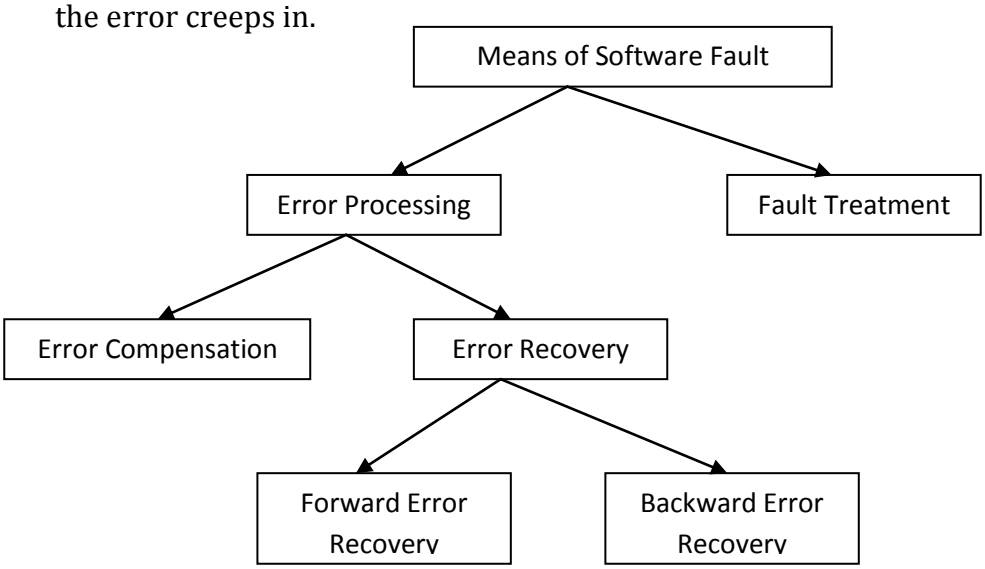
- ✓ Backward recovery: system brought back in state visited (Before error occurrence)
  Recovery points (checkpoint)
- ✓ Forward recovery: Erroneous state is discarded and correct one is determined without losing any computation.

Error processing aims to remove errors from the software state and can be implemented by substituting an error-free state in place of the erroneous state, called error recovery, or by compensating for the error by providing redundancy, called error compensation. Error recovery can be achieved by either forward or backward error recovery.

Fault Treatment: [23] avoiding fault(s) to be activated again

- ✓ **Fault Diagnosis:** Determination of error causes
- ✓ **Fault Isolation:** Removing faulty components from Subsequent execution process. System no longer able to deliver same service.
- ✓ **Reconfiguration:** Modification of system structure, such that non- Failed components deliver degraded service

Fault treatment, aims to prevent activation of faults and so action is taken before the error creeps in.



6. Design of Fault Tolerance:

A fault tolerance is a setup or configuration that prevents a computer or network device from failing in the event of an unexpected problem or error. To make a computer or network fault tolerant requires that the user or company to think how a computer or network device may fail and take steps that help prevent that type of failure. Below are some examples of steps that can be taken [10] [11].

A fault is a term that describes a malfunction that occurs in hardware or software.

- ✓ **Power Failure** - Have the computer or network device running on a UPS. In the event of a power outage, make sure the UPS can notify an administrator and properly turn off the computer after a few minutes if the power is not restored.
- ✓ **Power Surge** - If no UPS is connected to the computer or the UPS does not provide surge protection, a surge protector connected to the computer or network device would help prevent the device from failing in the event of a power surge.
- ✓ **Data loss** - Run backups daily or at least monthly on the computer if important information is stored on it. Create a mirror of the data on an alternate location.
- ✓ **Device or Computer failure -** Have a second device, computer, or computer components available in the event of failure to prevent a long down time.
- ✓ **Unauthorized access** - If connected to a network, setup a firewall.
- ✓ **Frequently check for updates** - Make sure the operating system and any running programs have the latest updates.
- ✓ **Lock device or password protect computer** - When not in use lock the computer and store the computer or network device in a secure area.
- ✓ **Overload** - Setup an alternate computer or network device that can be used as an alternative access point or can share the load either through a load balancing or round robin setup.
- ✓ **Virus** - Make sure the computer has updated virus definitions.

7. Fault-Tolerance Response Stages:

[12][13]The following table shows the detail of the ten system fault response stages, and gives each stage a detailed explanation and some more points that need to pay attention.

Fault Confinement: This technique may be applied in both hardware and software. For instance, it can be achieved by liberal use of fault detection circuits, consistency checks before performing a function ("mutual suspicion"), and multiple requests / confirmations before performing a function.

Fault Detection: Locate the fault. Multiple techniques have been developed and applied for fault detection. They can be basically classified into off-line fault detection and on-line fault detection. With the off-line detection, the device is unable to perform any function during the test, while for the on-line detection; the operation can keep going on while tests and the consequent work are being applied.

Fault Masking: Also called static redundancy, fault masking techniques hide the effects of failures through the means that redundant information outweighs the incorrect information. Majority voting is an example of fault masking.

Retry: It may appear that "retry" should be attempted after recovery is affected. But many times an operation that failed will execute correctly if it is tried again immediately. For instance, a transient error may prevent a successful operation, but an

immediate retry will succeed since the transient will have died away a few moments later.

Diagnosis: Diagnosis stage becomes necessary when detection could not provide fault location and other fault information

Reconfiguration: If a gut 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 as called graceful degradation

Recovery: After detection and maybe 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 recommences from this point. This form of recovery, often called rollback, usually entails strategies using backup files, check pointing, and journaling.

Restart: This might be possible in the case 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 the occurred damage is not unrecoverable. 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: Replace the damaged component. It can be either off-line or on-line.

Reintegration: After all, the repaired the device or module is reintegrated into the system. And especially for on-line repair, this has to be done without delay system operation.

8. Applications of Fault-Tolerance:

Fault-tolerance techniques were used to cope with physical defects of individual hardware components. [20-22]Designers of early computing systems employed redundant structures with voting to eliminate the effect of failed components; error-detection or correcting codes to detect or correct information errors, diagnostic techniques to locate failed components and automatic switch over's to replace them.

Fault tolerance can include:

- ✓ Responding to a power failure
- ✓ Immediately using a backup system in the event of a system failure
- ✓ Allowing mirrored disks to immediately take over for a failed disk

Multiple processors working together and comparing data and output for errors, then immediately correcting the detected errors.

9. Conclusion:

In this survey, a lot of techniques have been developed for achieving fault tolerance in software. The application of all of these techniques is relatively new to the area of fault tolerance. Furthermore, each technique will need to be suited to particular applications. [23] [24] The differences between each technique provide some flexibility of application. In general fault tolerance computing is considered as a study of faults, as mastering of faults behavior is the reasonable starting point of stopping their effects as any system defects. Most of the techniques and tools are generated initially for coping with hardware defects. And software fault tolerance research has drawn more and more focus nowadays, as the majority of system defects are shown to be software defects. Fault removal methods aim to remove faults after the development stage is completed. [25] This is done by exhaustive and acute testing of the final product. Fault tolerance method makes the assumption that the system has unavoidable and undetectable faults

and aims to make provisions for the system to operate correctly even in the presence of faults.

10. References

1. Chris Inacio "Software Fault Tolerance" Spring1998.
2. Ying Shi "Fault Tolerance Computing—Draft" Spring1999.
3. Hampton, Virginia "Software Fault Tolerance: A Tutorial"Wilfredo Torres-Pomales Langley Research Center NASA / TM-2000-210616.
4. P. Jalote "Fault Tolerance in Distributed Systems" Prentice-Hall, Englewood Cliffs, New Jersey, 1994. [bib]
5. Randell, B. "System Structure for Software Fault Tolerance" IEEE Transactions on Software Engineering, Vol. SE-1, No. 2, 1975, pp. 220.232.
6. "Software fault tolerance techniques and implementation" British Library Cataloguing in Publication Data Pullum, Laura ISBN 1-58053-470-8
7. "System Structure for Software Fault Tolerance" Brian Randell IEEE Transactions on Software Engineering, VOL. SE-1, NO. 2, JUNE 1975
8. Michael Ly "Software Fault Tolerance in Computer Operating Systems" Chapter 11 in Software Fault Tolerance (NASA-CR-197999)
9. P. J. Denning "Fault tolerant operating systems". ACM Computing Surveys ISSN 0360-0300 (December 1976).
10. Sarpreet Singh, Inderpreet Kauret al "Fault Tolerance Testing for Crash and Omission Transient Failure during Resource Scheduling of Grid Computing", International Journal of Computer Science and Mobile Computing, Vol.3 Issue.5, May- 2014.
11. ZaipengXie, Hongyu Sun and Kewal Saluja "A Survey of Software Fault Tolerance Techniques" *University of Wisconsin-Madison/Department of Electrical and Computer Engineering1415 Engineering Drive, Madison WI 53706 USA.
12. Algirdas Avizienis "Fault-Tolerant Systems" IEEE Transactions on Computers, VOL. C-25.
13. Avizienis, A. "Dependable Computing and Fault-Tolerant Systems" Vol. 1: TheEvolution of Fault-Tolerant Computing, Vienna: Springer-Verlag 1987.
14. Cott D. Stoller, Fred B. Schneider "Automated Analysis of Fault-Tolerancein Distributed Systems Formal Methods in System Design" 2005Springer Science + Business Media.
15. Elena Dubrova, "Fault Tolerant Design: An Introduction" Royal Institute of Technology Stockholm, Sweden Kluwer Academic Publishers Boston/Dordrecht/London
16. Kjetil Nørvag, "An Introduction to Fault-Tolerant Systems" Norwegian University of Science and Technology7034 Trondheim, Norway, IDI Technical Report 6/99, Revised July 2000ISSN 0802-6394.
17. Arif Sari, Murat Akkaya, "Fault Tolerance Mechanisms in Distributed Systems" Int. J. Communications, Network and System Sciences, 2015, 8, 471-482Published Online December 2015
18. R. K. Bawa Ramandeep Singh, "Comparative Analysis of Fault Tolerance Techniques in Grid Environment" International Journal of Computer Applications (0975 – 8887) Volume 41– No.1, March 2012
19. By Charlie Russel and Sharon Crawford "Planning Fault Tolerance and Avoidance" Chapter 7 from Microsoft Windows 2000 Server Administrator's Companion, published by Microsoft Press.

20. M. R. Lyu, ed., Software Fault Tolerance Chichester, England: John Wiley and Sons, Inc., 1995. The authoritative book on the subject of software fault tolerance written by the experts in the field.
21. P. Murray, R. Fleming, P. Harry, and P. Vickers, "Somersault Software Fault-Tolerance," HP Labs whitepaper, Palo Alto, California, 1998. An interesting paper on distributed rollback and recovery. It mentions an single interesting possibility of fault tolerance.
22. B. W. Johnson. Design and Analysis of Fault-Tolerant Digital Systems. Addison-Wesley, 1989.
23. J.C Laprie, Dependable computing and fault tolerance Concepts and terminology. In The 15th International Symposium on Fault-Tolerant Computing, pages 2–11, 1985.
24. A. Avizienis, "Architecture of fault-tolerant computing systems," in Dig. 1975 Int. Symp. Fault-Tolerant Computing, Paris, France, June 1975.
25. Y.-W. Ng and A. Aviiienis, "A model for transient and permanent fault recovery in closed fault tolerant systems," in Proc. 1976Int. Symp. Fault-Tolerant Computing, Pittsburgh, PA, June 1976, pp 182-188.