Why do we need a fault tolerant system?

Pouya Shams Ahari
Department of Electrical Engineering. Ahar Branch, Islamic Azad University, Ahar, Iran
pouyashams1@yahoo.com

Introduction

All designers try to get rid of all the hardware and software tools before they hit the market, but it shows that such a goal is unattainable. Some of these environmental factors are unexpected and unavoidable, as well as some potential mistakes are predictable. [2]Due to the development of semiconductor technologies, hardware components are naturally naturally reliable and the need for component fault tolerance in applications has been reduced. However, fault tolerance remains a requirement in many critical-safety, critical-mission, and critical-commercial applications.

Therefore, even when a system appears to be fully designed and implemented, the designer may experience errors outside of control. On the other hand, since it is practically impossible to build a complete and error-free system, fault tolerance is required. The main problem arises that he is sure he finds it, compensation can be applied. As a result, a tolerable system manages individual errors in hardware or software components, power supply failures, or other types of unforeseen adverse events.

What is fault tolerance

Fault tolerance refers to the ability of a system (computer, network, cloud cluster, etc.) to continue operating without interruption when one or more of its components fail.

The objective of creating a fault-tolerant system is to prevent disruptions arising from a single point of failure, ensuring the high availability and business continuity of mission-critical applications or systems.

Fault-tolerant systems use backup components that automatically take the place of failed components, ensuring no loss of service. [5]These include:

Hardware systems that are backed up by identical or equivalent systems. For example, a server can be made fault tolerant by using an identical server running in parallel, with all operations mirrored to the backup server.

Software systems that are backed up by other software instances. For example, a database with customer information can be continuously replicated to another machine. If the primary database goes down, operations can be automatically redirected to the second database.

Power sources that are made fault tolerant using alternative sources. For example, many organizations have power generators that can take over in case main line electricity fails.

In similar fashion, any system or component which is a single point of failure can be made fault tolerant using redundancy. [5] Fault tolerance can play a role in a disaster recovery strategy. For example, fault-

tolerant systems with backup components in the cloud can restore mission-critical systems quickly, even if a natural or human-induced disaster destroys on-premise IT infrastructure.

History

The first known fault-tolerant computer was SAPO, built in 1951 in Czechoslovakia by Antonín Svoboda.[3]:155 Its basic design was magnetic drums connected via relays, with a voting method of memory error detection (triple modular redundancy). Several other machines were developed along this line, mostly for military use. [4] Eventually, they separated into three distinct categories: machines that would last a long time without any maintenance, such as the ones used on NASA space probes and satellites; computers that were very dependable but required constant monitoring, such as those used to monitor and control nuclear power plants or supercollider experiments; and finally, computers with a high amount of runtime which would be under heavy use, such as many of the supercomputers used by insurance companies for their probability monitoring.

Most of the development in the so-called LLNM (Long Life, No Maintenance) computing was done by NASA during the 1960s, in preparation for Project Apollo and other research aspects. NASA's first machine went into a space observatory, and their second attempt, the JSTAR computer, was used in Voyager. This computer had a backup of memory arrays to use memory recovery methods and thus it was called the JPL Self-Testing-And-Repairing computer. [1]It could detect its own errors and fix them or bring up redundant modules as needed. The computer is still working today.

Hyper-dependable computers were pioneered mostly by aircraft manufacturers, nuclear power companies, and the railroad industry in the USA. These needed computers with massive amounts of uptime that would fail gracefully enough with a fault to allow continued operation while relying on the fact that the computer output would be constantly monitored by humans to detect faults. [2] Again, IBM developed the first computer of this kind for NASA for guidance of Saturn V rockets, but later on BNSF, Unisys, and General Electric built their own.

In the 1970s, much work has happened in the field . For instance, F14 CADC had built-in self-test and redundancy.

In general, the early efforts at fault-tolerant designs were focused mainly on internal diagnosis, where a fault would indicate something was failing and a worker could replace it. [2]SAPO, for instance, had a method by which faulty memory drums would emit a noise before failure. Later efforts showed that to be fully effective, the system had to be self-repairing and diagnosing – isolating a fault and then implementing a redundant backup while alerting a need for repair. This is known as N-model redundancy, where faults cause automatic fail-safes and a warning to the operator, and it is still the most common form of level one fault-tolerant design in use today.

Voting was another initial method, as discussed above, with multiple redundant backups operating constantly and checking each other's results, with the outcome that if,

for example, four components reported an answer of 5 and one component reported an answer of 6, the other four would "vote" that the fifth component was faulty and have it taken out of service. This is called M out of N majority voting.

Historically, the motion has always been to move further from N-model and more to M out of N due to the fact that the complexity of systems and the difficulty of ensuring the transitive state from fault-negative to fault-positive did not disrupt operations.

Tandem and Stratus were among the first companies specializing in the design of fault-tolerant computer systems for online transaction processing.

Fault tolerance vs. high availability

High availability refers to a system's ability to avoid loss of service by minimizing downtime. It's expressed in terms of a system's uptime, as a percentage of total running time.

[3]In most cases, a business continuity strategy will include both high availability and fault tolerance to ensure your organization maintains essential functions during minor failures, and in the event of a disaster.

While both fault tolerance and high availability refer to a system's functionality over time, there are differences that highlight their individual importance in your business continuity planning.

Consider the following analogy to better understand the difference between fault tolerance and high availability. A twin-engine airplane is a fault tolerant system – if one engine fails, the other one kicks in, allowing the plane to continue flying. Conversely, a car with a spare tire is highly available. A flat tire will cause the car to stop, but downtime is minimal because the tire can be easily replaced.

How to build a fault tolerance system?

In many applications, where a computer is used, interruptions or breakdowns can be costly or even catastrophic. In that case, the system has to deal with failures, but such systems are hardly complete.

The following is a summary of situations that may occur for any computer system, as well as safes that can help it work to some degree of acceptance if some of its components fail.

If something goes wrong, it will happen.

Murphy's First Law

There are countless ways in which a system can fail. In order to be able to tolerate the error, we must identify potential failures that a system may encounter and design countermeasures. [4]The frequency of each failure and its impact on the system must be estimated to decide which system to withstand. Here are just a few examples of potential issues to think about:

The application encounters an irreversible error and failure (uncontrolled exceptions, expired certificates, memory leaks)

Component not available (power failure, disconnection)

Data corruption (hardware failure, malicious attack)

Security (one component compromised)

Performance (increased latency, traffic, demand)

Most common failures can be divided into two categories:

Failure - Stop behaviors (eg server shutdown, connection loss)

Byzantine behaviors (such as data destruction or manipulation)

References

[1]. https://www.imperva.com/learn
[2]. Fault-Tolerant Design , Elena Dubrova (2013)
[3]. https://en.wikipedia.org/wiki/Fault_tolerance
[4]. Fault Tolerant System Design , Dr. Axel Krings (2011)
[5]. -https://kariera.future-processing.pl/blog