Availability

Availability refers to a property of software that it is there and ready to carry out its task when you need it to be. Availability builds upon the concept of reliability by adding the notion of recovery—that is, when the system breaks, it repairs itself.

The availability of a system can be calculated as the probability that it will provide the specified services within required bounds over a specified time interval. When referring to hardware, there is a well-known expression used to derive steady-state availability:

$$\frac{MTBF}{MTBF + MTTR}$$

where MTBF refers to the mean time between failures and MTTR refers to the mean time to repair.

Failure

A failure is the deviation of the system from its specification, where the deviation is externally visible. One of the most demanding tasks in building a high-availability, fault-tolerant system is to understand the nature of the failures that can arise during operation. Once those are understood, mitigation strategies can be designed into the software.

Fault

A failure's cause is called a fault. A fault can be either internal or external to the system under consideration.

Errors

Intermediate states between the occurrence of a fault and the occurrence of a failure are called errors.

PLANNING FOR FAILURES

1. **Hazard Analysis**
   Hazard analysis is a technique that attempts to catalog the hazards that can occur during the operation of a system. It categorizes each hazard according to its severity.
   For example, the DO-178B standard used in the aeronautics industry defines these failure condition levels in terms of their effects on the aircraft, crew, and passengers:
   - Catastrophic
   - Hazardous
   - Major
   - Minor
   - No effect

2. **Fault Tree Analysis**
   Fault tree analysis is an analytical technique that specifies a state of the system that negatively impacts safety or reliability, and then analyzes the system's context and operation to find all the ways that the undesired state could occur. The technique uses a graphic construct (the fault tree) that helps identify all sequential and parallel sequences of contributing faults that will result in the occurrence of the undesired state, which is listed at the top of the tree (the "top event"). The contributing faults might be hardware failures, human errors, software errors, or any other pertinent events that can lead to the undesired state.

Fault trees aid in system design, but they can also be used to diagnose failures at runtime. If the top event has occurred, then (assuming the fault tree model is complete) one or more of the contributing failures has occurred, and the fault tree can be used to track it down and initiate repairs.

AVAILABILITY GENERAL SCENARIO

From these considerations we can now describe the individual portions of an availability general scenario.

Source of stimulus:

We differentiate between internal and external origins of faults or failure because the desired system response may be different.

Stimulus:

A fault of one of the following classes occurs:

Omission: A component fails to respond to an input.

Crash: The component repeatedly suffers omission faults.

Timing: A component responds but the response is early or late.

Response: A component responds with an incorrect value.

Artifact:

This specifies the resource that is required to be highly available, such as a processor, communication channel, process, or storage.

Environment:

The state of the system when the fault or failure occurs may also affect the desired system response.

Response:

number of possible reactions to a system fault.

- Prevent the fault from becoming a failure
- Detect the fault:
    - Log the fault
    - Notify appropriate entities (people or systems)
- Recover from the fault:
    - Disable source of events causing the fault
    - Be temporarily unavailable while repair is being affected
    - Fix or mask the fault/failure or contain the damage it causes
    - Operate in a degraded mode while repair is being affected

Response measure:

The response measure can specify an availability percentage, or it can specify a time to detect the fault, time to repair the fault, times or time intervals during which the system must be available, or the duration for which the system must be available.

TACTICS FOR AVAILABILITY

Availability tactics may be categorized as addressing one of three categories: fault detection, fault recovery, and fault prevention.

DETECT FAULTS

Ping/echo refers to an asynchronous request/response message pair exchanged between nodes, used to determine reachability and the round-trip delay through the associated network path.

Monitor is a component that is used to monitor the state of health of various other parts of the system: processors, processes, I/O, memory, and so on.

Heartbeat is a fault detection mechanism that employs a periodic message exchange between a system monitor and a process being monitored.

Time stamp. This tactic is used to detect incorrect sequences of events, primarily in distributed message-passing systems. A time stamp of an event can be established by assigning the state of a local clock to the event immediately after the event occurs. Simple sequence numbers can also be used for this purpose, if time information is not important.

Sanity checking checks the validity or reasonableness of specific operations or outputs of a component. This tactic is typically based on a knowledge of the internal design, the state of the system, or the nature of the information under scrutiny.

Condition monitoring involves checking conditions in a process or device, or validating assumptions made during the design.

Voting. The most common realization of this tactic is referred to as triple modular redundancy (TMR), which employs three components that do the same thing, each of which receives identical inputs, and forwards their output to voting logic, used to detect any inconsistency among the three output states.

- *Replication* is the simplest form of voting; here, the components are exact clones of each other.
- Functional redundancy is a form of voting intended to address the issue of common-mode failures (design or implementation faults) in hardware or software components.
- Analytic redundancy permits not only diversity among components' private sides, but also diversity among the components' inputs and outputs.

Exception detection refers to the detection of a system condition that alters the normal flow of execution. The exception detection tactic can be further refined:

- *System exceptions* will vary according to the processor hardware architecture employed and include faults such as divide by zero, bus and address faults, illegal program instructions, and so forth.
- The *parameter fence* tactic incorporates an a priori data pattern (such as 0xDEADBEEF) placed immediately after any variable-length parameters of an object.
- Parameter typing employs a base class that defines functions that add, find, and iterate over type-length-value (TLV) formatted message parameters.
- *Timeout* is a tactic that raises an exception when a component detects that it or another component has failed to meet its timing constraints.

Self-test. Components (or, more likely, whole subsystems) can run procedures to test themselves for correct operation. Self-test procedures can be initiated by the component itself, or invoked from time to time by a system monitor.

RECOVER FROM FAULTS

Recover-from-faults tactics are refined into *preparation-and-repair tactics* and *reintroduction tactics*.

Preparation-and-repair tactics are based on a variety of combinations of retrying a computation or introducing redundancy. They include the following:

Active redundancy (hot spare). This refers to a configuration where all of the nodes (active or redundant spare) in a protection group2 receive and process identical inputs in parallel, allowing the redundant spare(s) to maintain synchronous state with the active node(s). Because the redundant spare possesses an identical state to the active processor, it can take over from a failed component in a matter of milliseconds.

Passive redundancy (warm spare). This refers to a configuration where only the active members of the protection group process input traffic; one of their duties is to provide the redundant spare(s) with periodic state updates.

Spare (cold spare). Cold sparing refers to a configuration where the redundant spares of a protection group remain out of service until a fail-over occurs, at which point a power-on-reset procedure is initiated on the redundant spare prior to its being placed in service.

Exception handling. Once an exception has been detected, the system must handle it in some fashion. The easiest thing it can do is simply to crash, but of course that's a terrible idea from the point of availability, usability, testability, and plain good sense.

Rollback. This tactic permits the system to revert to a previous known good state, referred to as the "rollback line"—rolling back time—upon the detection of a failure. Once the good state is reached, then execution can continue.

Software upgrade is another preparation-and-repair tactic whose goal is to achieve in-service upgrades to executable code images in a non-service-affecting manner. This may be realized as a function patch, a class patch, or a hitless in-service software upgrade (ISSU).

Retry. The retry tactic assumes that the fault that caused a failure is transient and retrying the operation may lead to success.

Ignore faulty behavior. This tactic calls for ignoring messages sent from a particular source when we determine that those messages are spurious

The degradation tactic maintains the most critical system functions in the presence of component failures, dropping less critical functions. This is done in circumstances where individual component failures gracefully reduce system functionality rather than causing a complete system failure.

Reconfiguration attempts to recover from component failures by reassigning responsibilities to the (potentially restricted) resources left functioning, while maintaining as much functionality as possible

Reintroduction is where a failed component is reintroduced after it has been corrected. Reintroduction tactics include the following:

The shadow tactic refers to operating a previously failed or in-service upgraded component in a "shadow mode" for a predefined duration of time prior to reverting the component back to an active role.

State resynchronization is a reintroduction partner to the active redundancy and passive redundancy preparation-and-repair tactics.

Escalating restart is a reintroduction tactic that allows the system to recover from faults by varying the granularity of the component(s) restarted and minimizing the level of service affected.

Non-stop forwarding (NSF) is a concept that originated in router design. In this design functionality is split into two parts: supervisory, or control plane (which manages connectivity and routing information), and data plane (which does the actual work of routing packets from sender to receiver).

PREVENT FAULTS

Prevent Faults Instead of detecting faults and then trying to recover from them, what if your system could prevent them from occurring in the first place? Although this sounds like some measure of clairvoyance might be required, it turns out that in many cases it is possible to do just that.

Removal from service. This tactic refers to temporarily placing a system component in an out-of-service state for the purpose of mitigating potential system failures. One example involves taking a component of a system out of service and resetting the component in order to scrub latent faults.

Transactions. Systems targeting high-availability services leverage transactional semantics to ensure that asynchronous messages exchanged between distributed components are atomic, consistent, isolated, and durable. These four properties are called the "ACID properties". This tactic prevents race conditions caused by two processes attempting to update the same data item.

Predictive model. A predictive model, when combined with a monitor, is employed to monitor the state of health of a system process to ensure that the system is operating within its nominal operating parameters, and to take corrective action when conditions are detected that are predictive of likely future faults.

Exception prevention. This tactic refers to techniques employed for the purpose of preventing system exceptions from occurring. The use of exception classes, which allows a system to transparently recover from system exceptions is one of the examples.

Increase competence set. A program's competence set is the set of states in which it is "competent" to operate. Increasing a component's competence set means designing it to handle more cases—faults—as part of its normal operation.