Module 2-CS 02
L02 Availability

BITS Pilani
Pilani|Dubai|Goa|Hyderabad

Harvinder S Jabbal
SSZG653 Software Architectures

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Module 2-CS 02
L02 Availability

Outline

What is Availability?

Availability General Scenario

Tactics for Availability

A Design Checklist for Availability

Summary

Credits

These Slides are based on

- Software Architecture in Practice by
  - Len Bass, Paul Clement and Rick Kazman
  - Pearson © 2013

What is 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.
- This is a broad perspective and encompasses what is normally called reliability.
- Availability builds on reliability by adding the notion of recovery (repair).
- Fundamentally, availability is about minimizing service outage time by mitigating faults.

Availability General Scenario

innovate achieve lead

Portion of Scenario	Possible Values
Source	Internal/external: people, hardware, software, physical infrastructure, physical environment
Stimulus	Fault: omission, crash, incorrect timing, incorrect response
Artifact	System's processors, communication channels, persistent storage, processes
Environment	Normal operation, startup, shutdown, repair mode, degraded operation, overloaded operation
Response	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 effected • fix or mask the fault/failure or contain the damage it causes • operate in a degraded mode while repair is being effected
Response Measure	Time or time interval when the system must be available Availability percentage (e.g. 99.999%) Time to detect the fault Time to repair the fault Time or time interval in which system can be in degraded mode Proportion (e.g., 99%) or rate (e.g., up to 100 per second) of a certain class of faults that the system prevents, or handles without failing

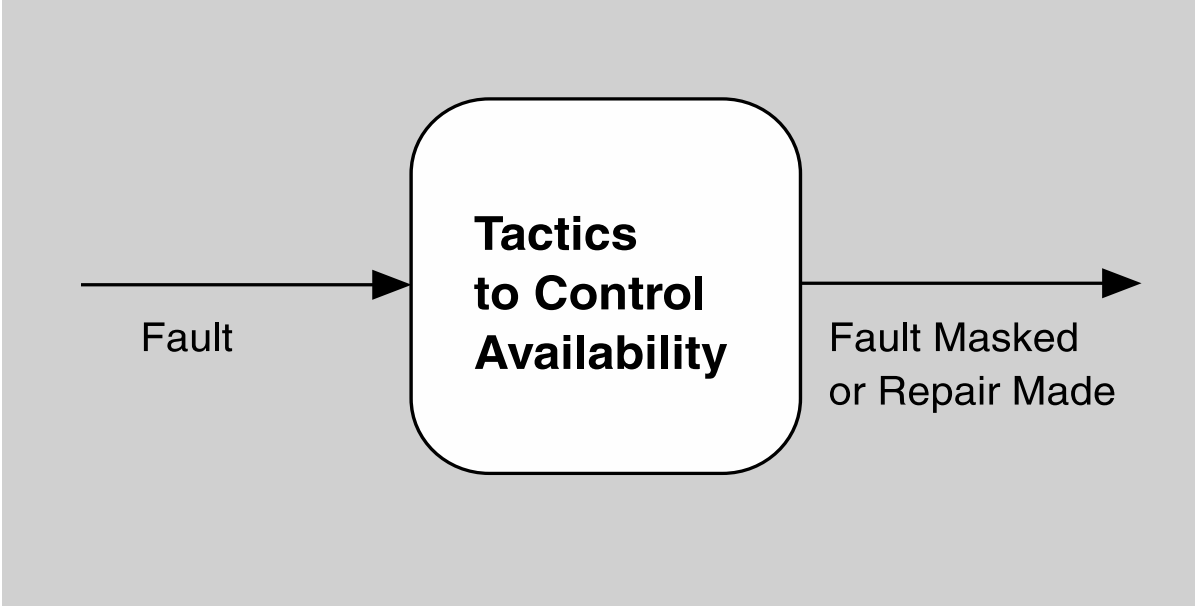
Sample Concrete Availability Scenario

The heartbeat monitor determines that the server is nonresponsive during normal operations. The system informs the operator and continues to operate with no downtime.

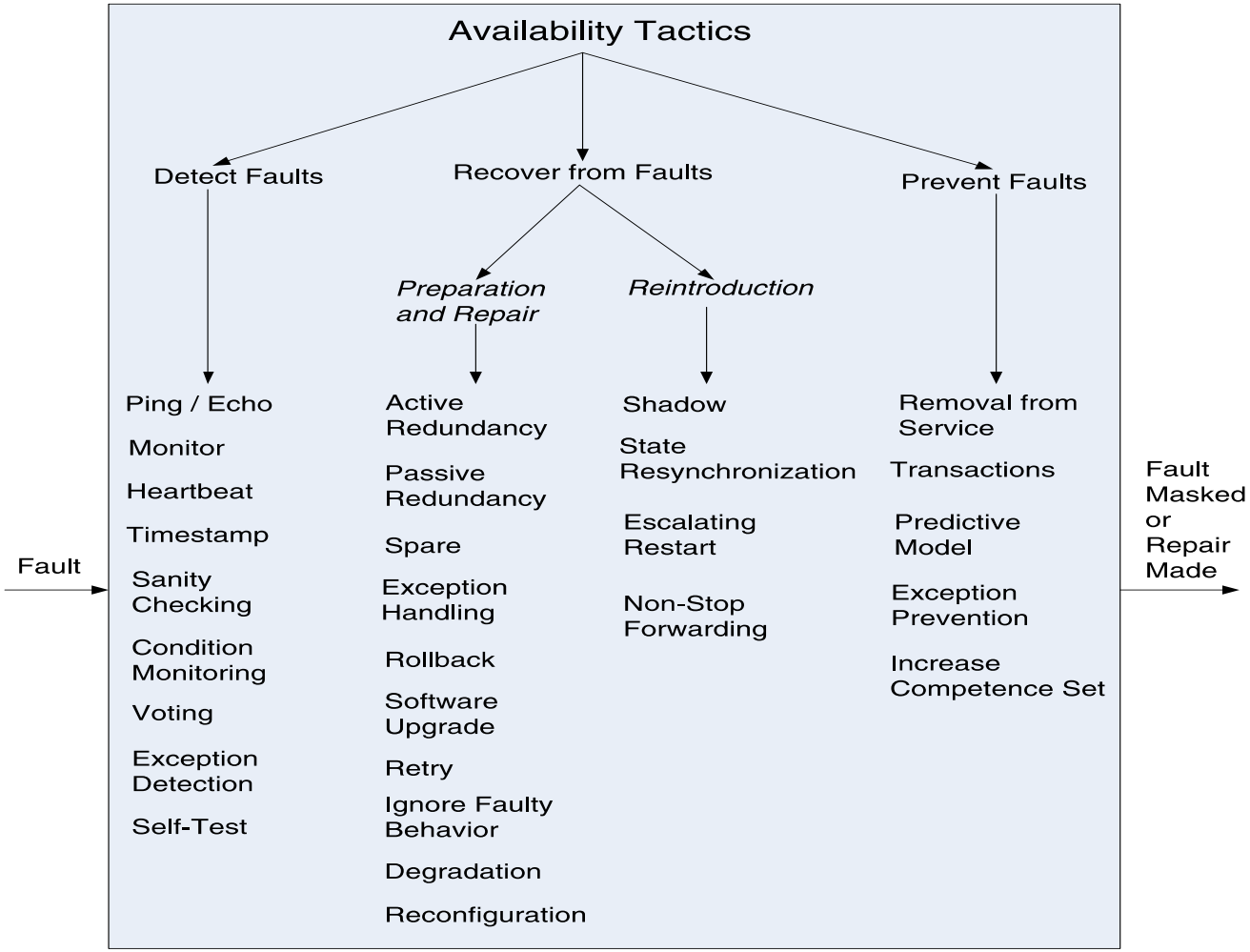
Goal of Availability Tactics

- A failure occurs when the system no longer delivers a service consistent with its specification
  - this failure is observable by the system's actors.
- A fault (or combination of faults) has the potential to cause a failure.
- Availability tactics enable a system to endure faults so that services remain compliant with their specifications.
- The tactics keep faults from becoming failures or at least bound the effects of the fault and make repair possible.

Goal of Availability Tactics



Availability Tactics



Detect Faults

- Ping/echo: asynchronous request/response message pair exchanged between nodes, used to determine reachability and the round-trip delay through the associated network path.
- Monitor: a component used to monitor the state of health of other parts of the system. A system monitor can detect failure or congestion in the network or other shared resources, such as from a denial-of-service attack.
- Heartbeat: a periodic message exchange between a system monitor and a process being monitored.

Detect Faults

innovate achieve lead

- Timestamp: used to detect incorrect sequences of events, primarily in distributed message-passing systems.
- Sanity Checking: checks the validity or reasonableness of a component's operations or outputs; typically based on a knowledge of the internal design, the state of the system, or the nature of the information under scrutiny.
- Condition Monitoring: checking conditions in a process or device, or validating assumptions made during the design.

Detect Faults

- Voting: to check that replicated components are producing the same results. Comes in various flavors: replication, functional redundancy, analytic redundancy.
- Exception Detection: detection of a system condition that alters the normal flow of execution, e.g. system exception, parameter fence, parameter typing, timeout.
- Self-test: procedure for a component to test itself for correct operation.

Recover from Faults (Preparation & Repair)

- Active Redundancy (hot spare): all nodes in a *protection group* receive and process identical inputs in parallel, allowing redundant spare(s) to maintain synchronous state with the active node(s).
  - A protection group is a group of nodes where one or more nodes are “active,” with the remainder serving as redundant spares.
- Passive Redundancy (warm spare): 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): 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.

Recover from Faults (Preparation & Repair)

- Exception Handling: dealing with the exception by reporting it or handling it, potentially masking the fault by correcting the cause of the exception and retrying.
- Rollback: revert to a previous known good state, referred to as the “rollback line”.
- Software Upgrade: in-service upgrades to executable code images in a non-service-affecting manner.

Recover from Faults (Preparation & Repair)

- Retry: where a failure is transient retrying the operation may lead to success.
- Ignore Faulty Behavior: ignoring messages sent from a source when it is determined that those messages are spurious.
- Degradation: maintains the most critical system functions in the presence of component failures, dropping less critical functions.
- Reconfiguration: reassigning responsibilities to the resources left functioning, while maintaining as much functionality as possible.

Recover from Faults (Reintroduction)

- Shadow: operating a previously failed or in-service upgraded component in a “shadow mode” for a predefined time prior to reverting the component back to an active role.
- State Resynchronization: partner to active redundancy and passive redundancy where state information is sent from active to standby components.
- Escalating Restart: recover from faults by varying the granularity of the component(s) restarted and minimizing the level of service affected.
- Non-stop Forwarding: functionality is split into supervisory and data. If a supervisor fails, a router continues forwarding packets along known routes while protocol information is recovered and validated.

Prevent Faults

- Removal From Service: temporarily placing a system component in an out-of-service state for the purpose of mitigating potential system failures
- Transactions: bundling state updates so that asynchronous messages exchanged between distributed components are *atomic*, *consistent*, *isolated*, and *durable*.
- Predictive Model: monitor the state of health of a process to ensure that the system is operating within nominal parameters; take corrective action when conditions are detected that are predictive of likely future faults.

Prevent Faults

- Exception Prevention: preventing system exceptions from occurring by masking a fault, or preventing it via smart pointers, abstract data types, wrappers.
- Increase Competence Set: designing a component to handle more cases—faults—as part of its normal operation.

Design Checklist for Availability

innovate achieve lead

Allocation of Responsibilities

- Determine the system responsibilities that need to be highly available. Ensure that additional responsibilities have been allocated to detect an omission, crash, incorrect timing, or incorrect response.
- Ensure that there are responsibilities to:
  - log the fault
  - notify appropriate entities (people or systems)
  - disable source of events causing the fault
  - be temporarily unavailable
  - fix or mask the fault/failure
  - operate in a degraded mode

Design Checklist for Availability

innovate achieve lead

Coordination Model

- Determine the system responsibilities that need to be highly available. With respect to those responsibilities
- Ensure that coordination mechanisms can detect an omission, crash, incorrect timing, or incorrect response. Consider, e.g., whether guaranteed delivery is necessary. Will the coordination work under degraded communication?
- Ensure that coordination mechanisms enable the logging of the fault, notification of appropriate entities, disabling of the source of the events causing the fault, fixing or masking the fault, or operating in a degraded mode
- Ensure that the coordination model supports the replacement of the artifacts (processors, communications channels, persistent storage, and processes). E.g., does replacement of a server allow the system to continue to operate?
- Determine if the coordination will work under conditions of degraded communication, at startup/shutdown, in repair mode, or under overloaded operation. E.g., how much lost information can the coordination model withstand and with what consequences?

Design Checklist for Availability

innovate achieve lead

Data Model

- Determine which portions of the system need to be highly available. Within those portions, determine which data abstractions could cause a fault of omission, a crash, incorrect timing behavior, or an incorrect response.
- For those data abstractions, operations, and properties, ensure that they can be disabled, be temporarily unavailable, or be fixed or masked in the event of a fault.
- E.g., ensure that write requests are cached if a server is temporarily unavailable and performed when the server is returned to service.

Design Checklist for Availability

innovate achieve lead

Mapping Among Architectural Elements

- Determine which artifacts (processors, communication channels, storage, processes) may produce a fault: omission, crash, incorrect timing, or incorrect response.
- Ensure that the mapping (or re-mapping) of architectural elements is flexible enough to permit the recovery from the fault. This may involve a consideration of
- which processes on failed processors need to be re-assigned at runtime
- which processors, data stores, or communication channels can be activated or re-assigned at runtime
- how data on failed processors or storage can be served by replacement units
- how quickly the system can be re-installed based on the units of delivery provided
- how to (re-) assign runtime elements to processors, communication channels, and data stores
- When employing tactics that depend on redundancy of functionality, the mapping from modules to redundant components is important. E.g., it is possible to write a module that contains code appropriate for both the active and back-up components in a protection group.

Design Checklist for Availability

innovate achieve lead

Resource Management

- Determine what critical resources are necessary to continue operating in the presence of a fault: omission, crash, incorrect timing, or incorrect response. Ensure there are sufficient remaining resources in the event of a fault to log the fault; notify appropriate entities (people or systems); disable source of events causing the fault; fix or mask the fault/failure; operate normally, in startup, shutdown, repair mode, degraded operation, and overloaded operation.
- Determine the availability time for critical resources, what critical resources must be available during specified time intervals, time intervals during which the critical resources may be in a degraded mode, and repair time for critical resources. Ensure that the critical resources are available during these time intervals.
- For example, ensure that input queues are large enough to buffer anticipated messages if a server fails so that the messages are not permanently lost.

Design Checklist for Availability

innovate achieve lead

Binding Time

- Determine how and when architectural elements are bound. If late binding is used to alternate between components that can themselves be sources of faults (e.g. processes, processors, communication channels), ensure the chosen availability strategy is sufficient to cover faults introduced by all sources. E.g.
- If late binding is used to switch between processors that will be the subject of faults, will the fault detection and recovery mechanisms work for all possible bindings?
- If late binding is used to change the definition or tolerance of what constitutes a fault (e.g., how long a process can go without responding before a fault is assumed), is the recovery strategy chosen sufficient to handle all cases? For example, if a fault is flagged after 0.1 ms, but the recovery mechanism takes 1.5 seconds to work, that might be an unacceptable mismatch.
- What are the availability characteristics of the late binding mechanism itself? Can it fail?

Design Checklist for Availability

Choice of Technology

- Determine the available technologies that can (help) detect faults, recover from faults, re-introduce failed components.
- Determine what technologies are available that help the response to a fault (e.g., event loggers).
- Determine the availability characteristics of chosen technologies themselves: What faults can they recover from? What faults might they introduce into the system?

Summary

- Availability refers to the ability of the system to be available for use when a fault occurs.
- The fault must be recognized (or prevented) and then the system must respond.
- The response will depend on the criticality of the application and the type of fault
  - can range from “ignore it” to “keep on going as if it didn’t occur.”

Summary

- Tactics for availability are categorized into detect faults, recover from faults and prevent faults.
- Detection tactics depend on detecting signs of life from various components.
- Recovery tactics are retrying an operation or maintaining redundant data or computations.
- Prevention tactics depend on removing elements from service or limiting the scope of faults.
- All availability tactics involve the coordination model.