5. Disponibilidad

Con james scott

El noventa por ciento de la vida está apareciendo.
-Woody Allen

La disponibilidad se refiere a una propiedad de software que está allí y lista para llevar a cabo su tarea cuando la necesita. Esta es una perspectiva amplia y abarca lo que normalmente se llama confiabilidad (aunque puede abarcar consideraciones adicionales como el tiempo de inactividad debido al mantenimiento periódico). De hecho, la disponibilidad se basa en el concepto de confiabilidad al agregar la noción de recuperación, es decir, cuando el sistema se rompe, se repara a sí mismo. La reparación puede realizarse por varios medios, que veremos en este capítulo. Más precisamente, Avižienis y sus colegas han definido la fiabilidad:

La confiabilidad es la capacidad de evitar fallas que son más frecuentes y más graves de lo que es aceptable.

Nuestra definición de disponibilidad como un aspecto de confiabilidad es la siguiente: "La disponibilidad se refiere a la capacidad de un sistema para enmascarar o reparar fallas, de modo que el período de interrupción acumulativa del servicio no exceda un valor requerido durante un intervalo de tiempo específico". Concepto de falla sujeto al juicio de un agente externo, posiblemente un humano. También incluyen conceptos de confiabilidad, confidencialidad, integridad y cualquier otro atributo de calidad que implique un concepto de falla inaceptable.

La disponibilidad está estrechamente relacionada con la seguridad. Un ataque de denegación de servicio está diseñado explícitamente para hacer que un sistema falle, es decir, para que no esté disponible. La disponibilidad también está estrechamente relacionada con el rendimiento, ya que puede ser difícil saber cuándo un sistema ha fallado y cuando simplemente está siendo escandalosamente lento para responder. Finalmente, la disponibilidad está estrechamente relacionada con la seguridad, que se refiere a mantenerel sistema no puede entrar en un estado peligroso y recuperar o limitar el daño cuando lo hace.

Fundamentalmente, la disponibilidad se trata de minimizar el tiempo de interrupción del servicio mediante la mitigación de fallas. El fracaso implica visibilidad para un sistema u observador humano en el entorno. Es decir, un fallo es la desviación del sistema de su especificación, donde la desviación es visible externamente. Una de las tareas más exigentes en la construcción de un sistema de alta disponibilidad y tolerante a fallas es comprender la naturaleza de las fallas que pueden surgir durante la operación (consulte la barra lateral " Planificación de fallas "). Una vez que se entienden, se pueden diseñar estrategias de mitigación en el software.

La causa de una falla se llama falla. Una falla puede ser interna o externa al sistema bajo consideración. Los estados intermedios entre la ocurrencia de una falla y la ocurrencia de una falla se llaman errores. Las fallas se pueden prevenir, tolerar, eliminar o pronosticar. De esta manera, un sistema se vuelve "resistente" a las fallas.

Entre las áreas que nos preocupan está la forma en que se detectan las fallas del sistema, la frecuencia con que pueden ocurrir las fallas del sistema, lo que sucede cuando ocurre una falla, el tiempo que un sistema puede estar fuera de servicio, cuando las fallas o fallas pueden ocurrir de manera segura, cómo se pueden prevenir las fallas o fallas, y qué tipo de notificaciones se requieren cuando ocurre una falla.

Debido a que los usuarios pueden observar una falla del sistema, el tiempo de reparación es el tiempo hasta que la falla deja de ser observable. Esto puede ser una breve demora en el tiempo de respuesta o puede ser el tiempo que le toma a alguien volar a un lugar remoto en los Andes para reparar una pieza de maquinaria para la minería (tal como nos lo contó una persona responsable de reparar el software en un motor de maquina minera). La noción de "observabilidad" puede ser complicada: el virus Stuxnet, como ejemplo, pasó desapercibido durante mucho tiempo a pesar de que estaba haciendo daño. Además, a menudo nos preocupa el nivel de capacidad que permanece cuando se produce una falla: un modo de operación degradado.

La distinción entre fallas y fallas permite la discusión de estrategias de reparación automática. Es decir, si se ejecuta el código que contiene una falla pero el sistema puede recuperarse de la falla sin que se observe ninguna desviación de un comportamiento específico, no hay falla.

La disponibilidad de un sistema se puede calcular como la probabilidad de que proporcionará los servicios especificados dentro de los límites requeridos durante un intervalo de tiempo especificado. Cuando se hace referencia al hardware, hay una expresión conocida que se usa para derivar la disponibilidad en estado estable:

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

donde *MTBF se* refiere al tiempo medio entre fallas y *MTTR se* refiere al tiempo medio para reparar. En el mundo del software, esta fórmula debe interpretarse en el sentido de que, al pensar en la disponibilidad, debe pensar en qué hará que su sistema falle, qué probabilidades hay de que ocurra y que se necesitará algo de tiempo para repararlo.

A partir de esta fórmula, es posible calcular probabilidades y hacer afirmaciones como "99.999 por ciento de disponibilidad", o un 0.001 por ciento de probabilidad de que el sistema no esté operativo cuando sea necesario. Los tiempos de inactividad programados (cuando el sistema se deja fuera de servicio intencionalmente) pueden no considerarse al calcular la disponibilidad, porque el sistema se considera "no necesario" en ese momento; Por supuesto, esto depende de los requisitos específicos para el sistema, a menudo codificados en acuerdos de nivel de servicio (SLA). Esta disposición puede dar lugar a situaciones aparentemente extrañas en las que el sistema está inactivo y los usuarios lo están esperando, pero el tiempo de inactividad está programado y, por lo tanto, no se incluye en los requisitos de disponibilidad.

En los sistemas operativos, las fallas se detectan y se correlacionan antes de ser reportadas y reparadas. La lógica de correlación de fallas categorizará una falla de acuerdo con su gravedad (crítica, mayor o menor) y el impacto del servicio (que afecta al servicio o que no afecta al servicio) para proporcionar al operador del sistema un estado oportuno y preciso del sistema y permitir la Estrategia de reparación adecuada para ser empleado. La estrategia de reparación puede ser automatizada o puede requerir intervención manual.

La disponibilidad proporcionada por un sistema informático o servicio de alojamiento se expresa frecuentemente como un acuerdo de nivel de servicio. Este SLA especifica el nivel de disponibilidad garantizado y, por lo general, las penalizaciones que sufrirá el sistema informático o el servicio de alojamiento si se infringe el SLA. El SLA que Amazon proporciona para su servicio de nube EC2 es

AWS realizará esfuerzos comercialmente razonables para que Amazon EC2 esté disponible con un Porcentaje de tiempo de actividad anual [definido en otra parte] de al menos 99.95% durante el Año de servicio. En el caso de que Amazon EC2 no cumpla con el compromiso de porcentaje de tiempo de actividad anual, será elegible para recibir un Crédito de servicio como se describe a continuación.

La Tabla 5.1 proporciona ejemplos de los requisitos de disponibilidad del sistema y los valores de umbral asociados para un tiempo de inactividad aceptable del sistema, medido en períodos de observación de 90 días y un año. El término *alta disponibilidad* generalmente se refiere a los diseños que apuntan a una disponibilidad del 99,999 por ciento ("5 nueves") o más. Por definición o convención, solo las interrupciones no programadas contribuyen al tiempo de inactividad del sistema.

Tabla 5.1. Requisitos de disponibilidad del sistema

Availability	Downtime/90 Days	Downtime/Year
99.0%	21 hours, 36 minutes	3 days, 15.6 hours
99.9%	2 hours, 10 minutes	8 hours, 0 minutes, 46 seconds
99.99%	12 minutes, 58 seconds	52 minutes, 34 seconds
99.999%	1 minute, 18 seconds	5 minutes, 15 seconds
99.9999%	8 seconds	32 seconds

Planificación para el fracaso

Al diseñar un sistema de alta disponibilidad o crítico para la seguridad, es tentador decir que la falla no es una opción. Es una frase pegadiza, pero es una pésima filosofía de diseño. De hecho, el fracaso no es solo una opción, es casi inevitable. Lo que hará que su sistema sea seguro y esté disponible es la planificación de la aparición de fallos o (más probablemente) fallos, y su manejo con aplomo. El primer paso es comprender qué tipo de fallas es propenso a su sistema y cuáles serán las consecuencias de cada una. Aquí hay tres técnicas bien conocidas para controlar esto.

Análisis de riesgo

El análisis de peligros es una técnica que intenta catalogar los peligros que pueden ocurrir durante la operación de un sistema. Se categoriza cada peligro de acuerdo a su severidad. Por ejemplo, el estándar DO-178B utilizado en la industria aeronáutica define estos niveles de condición de falla en términos de sus efectos en la aeronave, la tripulación y los pasajeros:

• *Catastrófico*. Este tipo de falla puede causar un accidente. Esta falla representa la pérdida de la función crítica requerida para volar y aterrizar de manera segura.

• *Peligroso*. Este tipo de falla tiene un gran impacto negativo en la seguridad o el rendimiento, o reduce la capacidad de la tripulación para operar la aeronave debido a una aflicción física o una mayor carga de trabajo, o causa lesiones graves o fatales entre los pasajeros.

• *Mayor* Este tipo de falla es importante, pero tiene un impacto menor que una falla peligrosa (por ejemplo, provoca molestias al pasajero en lugar de lesiones) o aumenta significativamente la carga de trabajo de la tripulación hasta el punto donde se ve afectada la seguridad.

• *Menores*. Este tipo de falla es notable, pero tiene un impacto menor que una falla mayor (por ejemplo, causando inconvenientes al pasajero o un cambio de rutina en el plan de vuelo).

• *Sin efecto*. Este tipo de falla no tiene impacto en la seguridad, la operación de la aeronave o la carga de trabajo de la tripulación.

Otros dominios tienen sus propias categorías y definiciones. El análisis de riesgos también evalúa la probabilidad de que ocurra cada peligro. Los peligros para los cuales el producto del costo y la probabilidad exceden algún umbral se convierten en el tema de las actividades de mitigación.

Análisis del árbol de fallos

El análisis del árbol de fallas es una técnica analítica que especifica un estado del sistema que afecta negativamente a la seguridad o confiabilidad, y luego analiza el contexto y la operación del sistema para encontrar todas las formas en que podría ocurrir el estado no deseado. La técnica utiliza una construcción gráfica (el árbol de fallas) que ayuda a identificar todas las secuencias secuenciales y paralelas de fallas contribuyentes que resultarán en la aparición del estado no deseado, que se encuentra en la parte superior del árbol (el "evento superior"). Las fallas que contribuyen pueden ser fallas de hardware, errores humanos, errores de software o cualquier otro evento pertinente que pueda llevar al estado no deseado.

La Figura 5.1 , tomada de un manual de la NASA sobre el análisis del árbol de fallas, muestra un árbol de fallas muy simple para el cual el evento principal es la falla del componente D. Muestra que la componente D puede fallar si A falla *y* B o C falla.

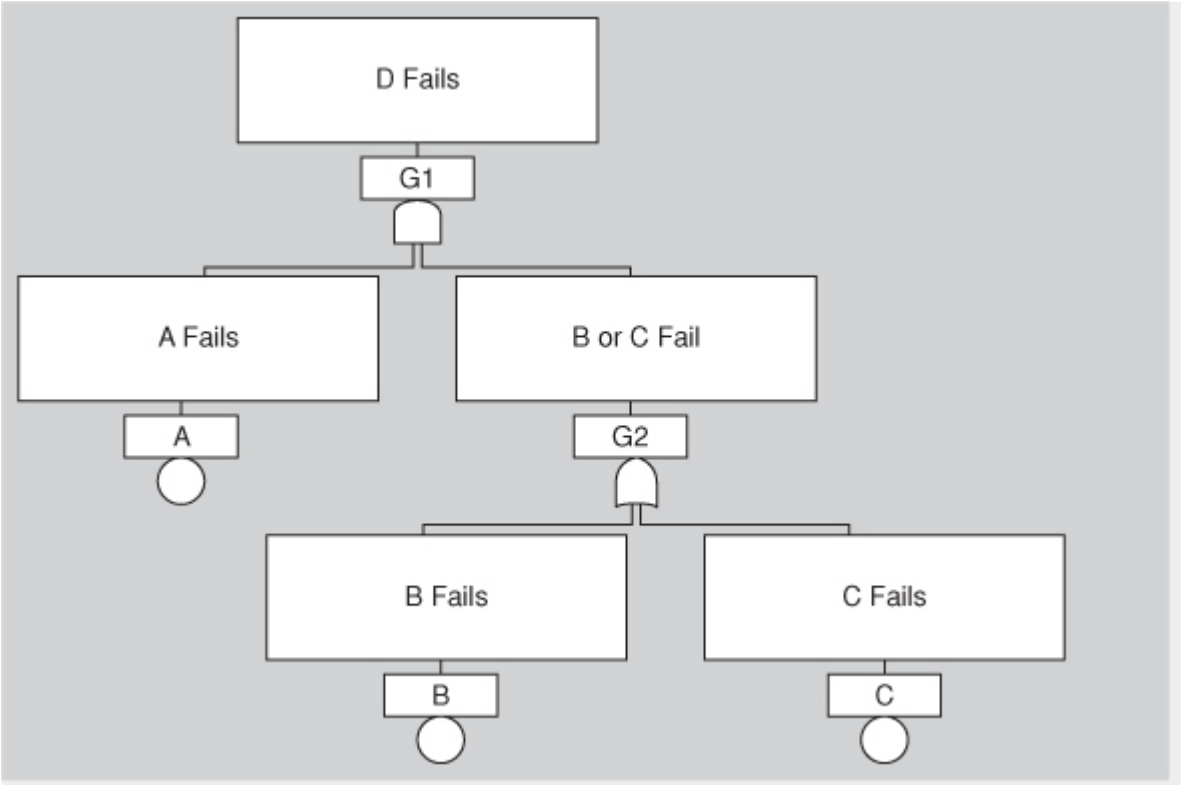

Figura 5.1. Un simple árbol de fallas. D falla si A falla y B o C falla.

Los símbolos que conectan los eventos en un árbol de fallas se denominan símbolos de puerta y se toman de diagramas lógicos booleanos. La figura 5.2 ilustra la notación.

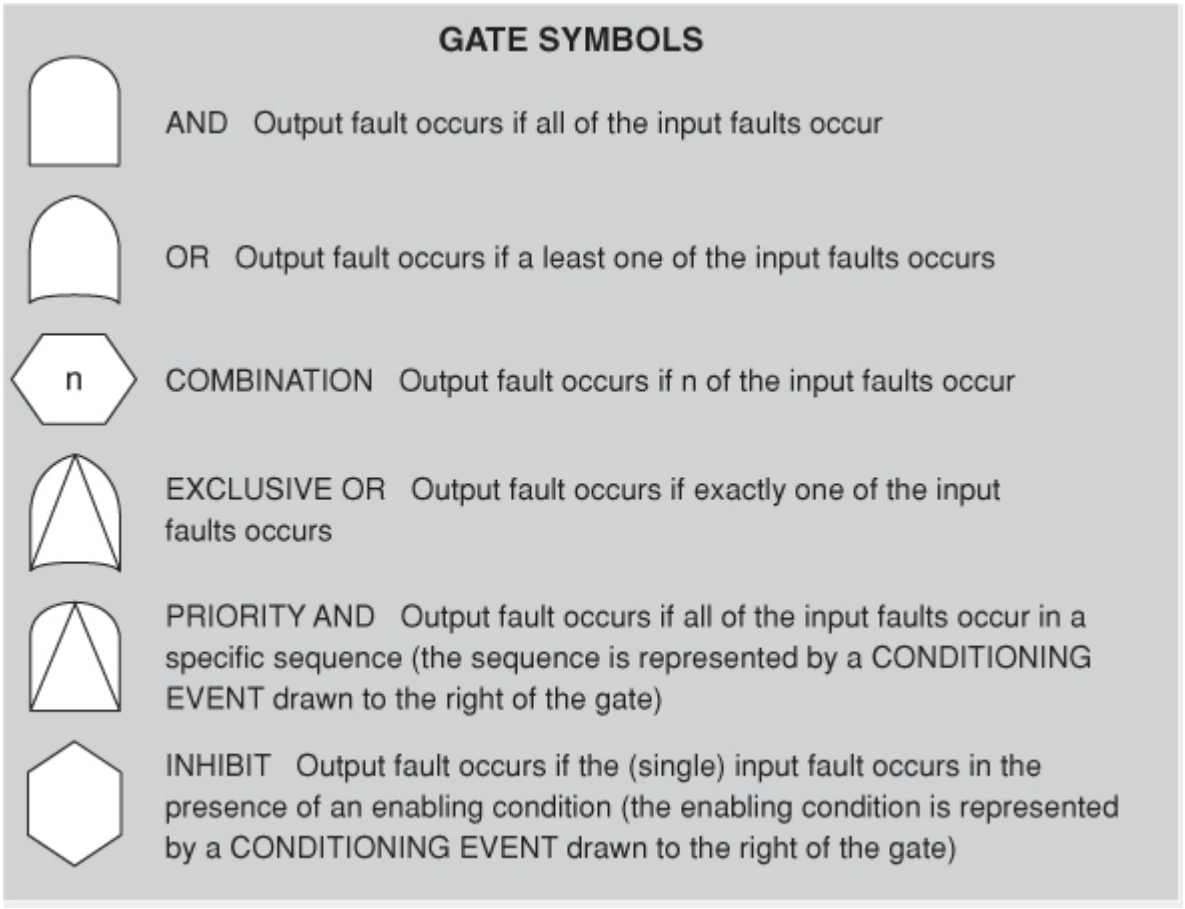

Figura 5.2. Símbolos de la puerta del árbol de fallas

Un árbol de fallas se presta al análisis estático de varias maneras. Por ejemplo, un "conjunto mínimo de cortes" es la combinación más pequeña de eventos a lo largo de la parte inferior del árbol que juntos pueden causar el evento principal. El conjunto de conjuntos de cortes mínimos muestra todas las formas en que los eventos de la parte inferior se pueden combinar para provocar el fallo general. Cualquier conjunto de cortes mínimos singleton revela un punto único de falla, que debe ser cuidadosamente analizado. Además, las probabilidades de varias fallas contribuyentes se pueden combinar para generar una probabilidad de que ocurra el evento principal. El análisis dinámico ocurre cuando el orden de las fallas que contribuyen importa. En este caso, se pueden usar técnicas como el análisis de Markov para calcular la probabilidad de falla en diferentes secuencias de falla.

Los árboles de fallas ayudan en el diseño del sistema, pero también pueden usarse para diagnosticar fallas en tiempo de ejecución. Si se produjo el evento principal, entonces (asumiendo que el modelo del árbol de fallas está completo), se han producido una o más de las fallas contribuyentes, y se puede usar el árbol de fallas para rastrearlo e iniciar las reparaciones.

El modo de fallas, los efectos y el análisis de criticidad (FMECA) catalogan los tipos de fallas que los sistemas de un tipo dado son propensos, junto con la gravedad de los efectos de cada uno. FMECA se basa en la historia de fallas de sistemas similares en el pasado. La Tabla 5.2 , también tomada del manual de la NASA, muestra los datos de un sistema de amplificadores redundantes. Los datos históricos muestran que los amplificadores fallan con mayor frecuencia cuando hay un cortocircuito o el circuito se deja abierto, pero también hay otros modos de falla (agrupados como "Otros").

Tabla 5.2. Probabilidades de falla y efectos

Component	Failure Probability	Failure Mode	% Failures by Mode	Effects	
				Critical	Noncritical
		Open	90		X
A	1×10^{-3}	Short	5	X (5×10^{-5})	
		Other	5	X (5×10^{-5})	
B	1×10^{-3}	Open	90		X
		Short	5	X (5×10^{-5})	
		Other	5	X (5×10^{-5})	

Agregar la columna crítica nos da la probabilidad de un fallo crítico del sistema: $5 \times 10^{-5} + 5 \times 10^{-5} + 5 \times 10^{-5} + 5 \times 10^{-5} = 2 \times 10^{-4}$.

Estas y otras técnicas son tan buenas como el conocimiento y la experiencia de las personas que pueblan sus respectivas estructuras de datos. Uno de los peores errores que puede cometer, según el manual de la NASA, es dejar que la forma tome prioridad sobre la sustancia. Es decir, no permita que la ingeniería de seguridad se convierta en una cuestión de simplemente llenar las tablas. En su lugar, siga presionando para averiguar qué más puede salir mal y luego planifíquelo.

5.1. DISPONIBILIDAD GENERAL ESCENARIO

A partir de estas consideraciones, ahora podemos describir las partes individuales de un escenario general de disponibilidad. Estos se resumen en la Tabla 5.3 :

• *Fuente de estímulo* . Diferenciamos entre orígenes internos y externos de fallas o fallas porque la respuesta deseada del sistema puede ser diferente.

• *Estímulo* . Se produce un fallo de una de las siguientes clases:

• *Omisión* . Un componente no responde a una entrada.

• *Crash* . El componente sufre repetidamente fallas de omisión.

• *Tiempo* . Un componente responde pero la respuesta es temprana o tardía.

• *Respuesta* . Un componente responde con un valor incorrecto.

• *Artefacto*. Esto especifica el recurso que se requiere que esté altamente disponible, como un procesador, canal de comunicación, proceso o almacenamiento.

• *Medio ambiente* . El estado del sistema cuando ocurre la falla o falla también puede afectar la respuesta del sistema deseada. Por ejemplo, si el sistema ya ha visto algunas fallas y está funcionando en un modo diferente al normal, puede ser conveniente apagarlo por completo. Sin embargo, si este es el primer fallo observado, puede preferirse alguna degradación del tiempo de respuesta o función.

• *Respuesta* . Hay una serie de posibles reacciones a una falla del sistema. Primero, la falla debe ser detectada y aislada (correlacionada) antes de que cualquier otra respuesta sea posible. (Una excepción a esto es cuando la falla se previene antes de que ocurra). Después de que se detecte la falla, el sistema debe recuperarse de ella. Las acciones asociadas con estas posibilidades incluyen el registro de la falla, notificar a los usuarios seleccionados u otros sistemas, tomar medidas para limitar el daño causado por la falla, cambiar a un modo degradado con menos capacidad o menos función, apagar sistemas externos o dejar de estar disponible durante reparar.

• *Medida de respuesta* . La medida de respuesta puede especificar un porcentaje de disponibilidad, o puede especificar un tiempo para detectar la falla, tiempo para reparar la falla, tiempos o intervalos de tiempo durante los cuales el sistema debe estar disponible, o la duración por la cual el sistema debe estar disponible.

Tabla 5.3. Disponibilidad General Escenario

Portion of Scenario	Possible Values
Source	Internal/external: people, hardware, software, physical infrastructure, physical environment
Stimulus	Fault: omission, crash, incorrect timing, incorrect response
Artifact	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

La Figura 5.3 muestra un escenario concreto generado a partir del escenario general: el monitor de latido determina que el servidor no responde durante las operaciones normales. El sistema informa al operador y continúa funcionando sin tiempo de inactividad.

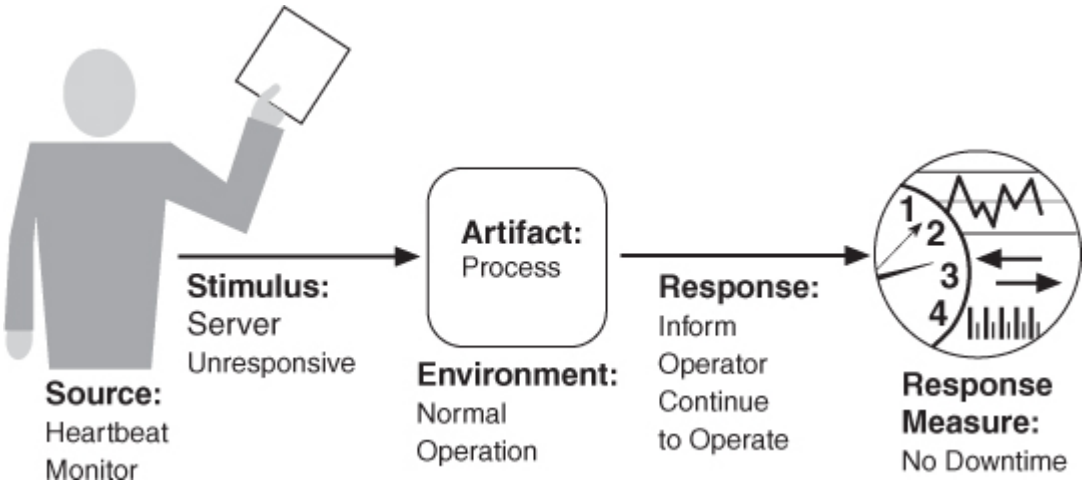

Figura 5.3. Ejemplo de escenario concreto de disponibilidad.

5.2. TÁCTICAS DE DISPONIBILIDAD

Se produce una falla cuando el sistema ya no entrega un servicio que sea consistente con su especificación; Este fallo es observable por los actores del sistema. Una falla (o combinación de fallas) tiene el potencial de causar una falla. Las tácticas de disponibilidad, por lo tanto, están diseñadas para permitir que un sistema sufra fallas en el sistema, de modo que un servicio que está siendo entregado por el sistema sigue cumpliendo con sus especificaciones. Las tácticas que discutimos en esta sección evitarán que las fallas se conviertan en fallas o, al menos, limiten los efectos de la falla y hagan posible la reparación. Ilustramos este enfoque en la figura 5.4 .

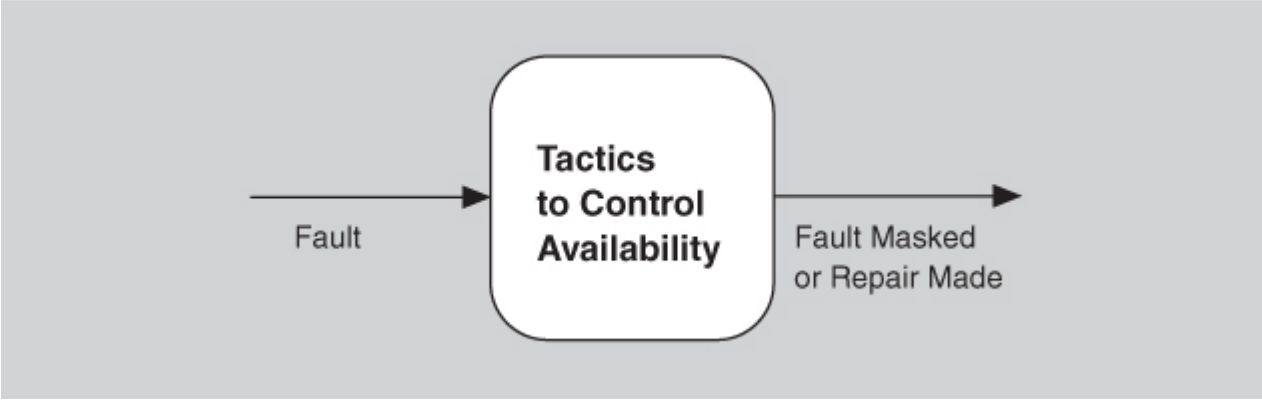

Figura 5.4. Objetivo de tácticas de disponibilidad.

Las tácticas de disponibilidad se pueden categorizar para abordar una de tres categorías: detección de fallas, recuperación de fallas y prevención de fallas. La categorización de tácticas para la disponibilidad se muestra en la Figura 5.5 (en la página siguiente). Tenga en cuenta que a menudo es el caso que estas tácticas le serán proporcionadas por una infraestructura de software, como un paquete de middleware, por lo que su trabajo como arquitecto suele consistir en elegir y evaluar (en lugar de implementar) las tácticas de disponibilidad correctas y la Combinación correcta de tácticas.

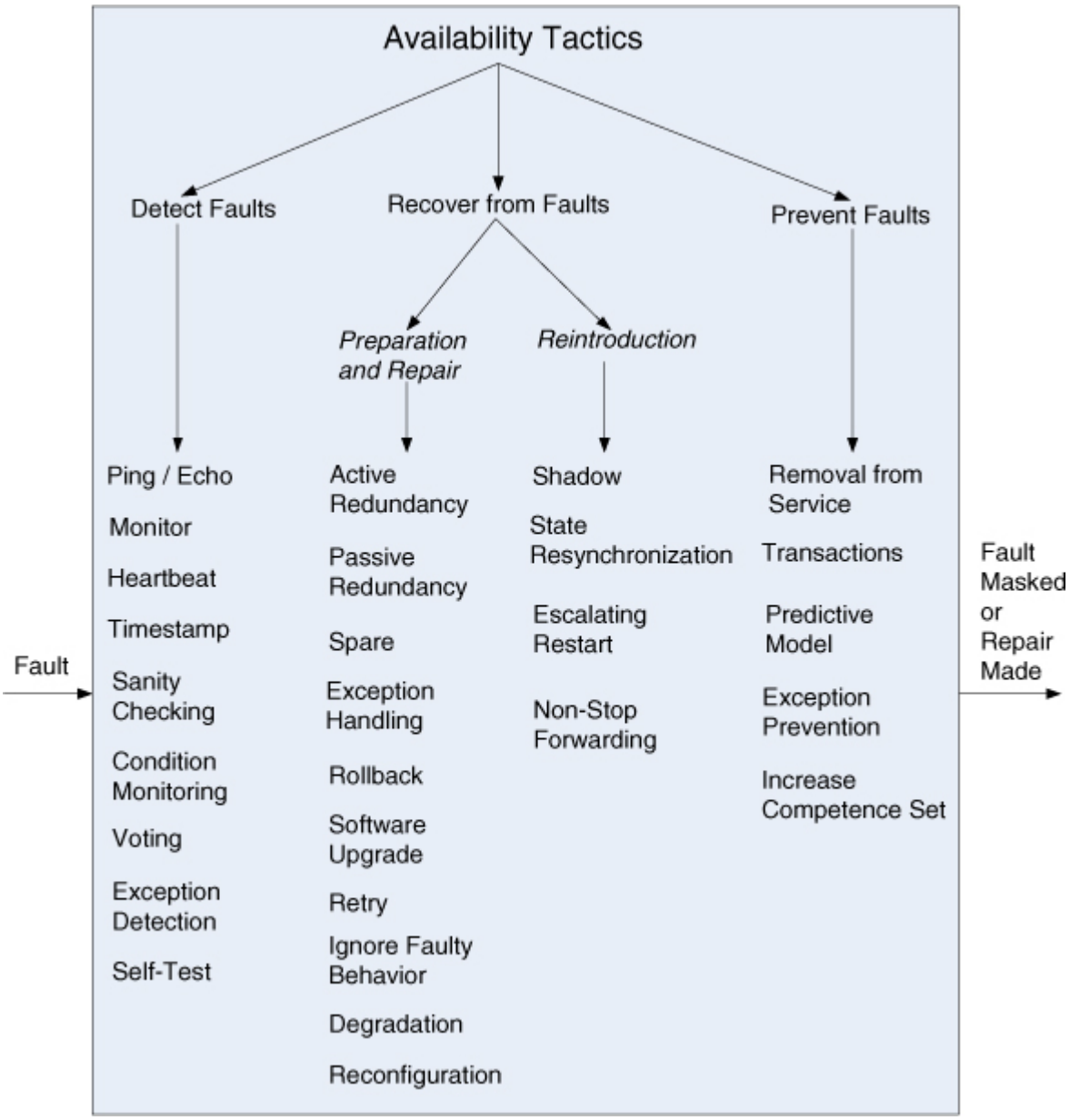

Figura 5.5. Tácticas de disponibilidad

Detectar fallas

Antes de que cualquier sistema pueda tomar medidas con respecto a una falla, la presencia de la falla debe ser detectada o anticipada. Las tácticas en esta categoría incluyen lo siguiente:

• *Ping / eco se* refiere a un par de mensajes de solicitud / respuesta asíncronos intercambiados entre nodos, utilizados para determinar la accesibilidad y el retardo de ida y vuelta a través de la ruta de red asociada. Pero el eco también determina que el componente ping está

vivo y responde correctamente. El ping esA menudo enviado por un monitor del sistema. Ping / echo requiere que se establezca un umbral de tiempo; este umbral le dice al componente de ping cuánto tiempo debe esperar el eco antes de considerar que el componente de ping ha fallado ("tiempo de espera"). Las implementaciones estándar de ping / echo están disponibles para nodos interconectados a través de IP.

• *Monitor* . Un monitor es un componente que se usa para monitorear el estado de salud de varias otras partes del sistema: procesadores, procesos, E / S, memoria, etc. Un monitor del sistema puede detectar fallas o congestiones en la red u otros recursos compartidos, como por ejemplo un ataque de denegación de servicio. Organiza el software utilizando otras tácticas en esta categoría para detectar componentes que no funcionan correctamente. Por ejemplo, el monitor del sistema puede iniciar las autocomprobaciones o ser el componente que detecta marcas de tiempo defectuosas o latidos cardíacos perdidos. [1]

• *Heartbeat* es un mecanismo de detección de fallas que emplea un intercambio periódico de mensajes entre un monitor del sistema y un proceso que se está monitoreando. Un caso especial de latidos cardíacos es cuando el proceso que se monitorea periódicamente restablece el temporizador de vigilancia en su monitor para evitar que caduque y, por lo tanto, señala un fallo. Para los sistemas en los que la escalabilidad es un problema, la sobrecarga de transporte y procesamiento puede reducirse mediante la combinación de mensajes de latido en otros mensajes de control que se intercambian entre el proceso que se monitorea y el controlador del sistema distribuido. La gran diferencia entre Heartbeat y ping / echo es quién tiene la responsabilidad de iniciar el control de estado: el monitor o el componente en sí.

• *Marca de tiempo*. Esta táctica se usa para detectar secuencias incorrectas de eventos, principalmente en sistemas de paso de mensajes distribuidos. Se puede establecer una marca de tiempo de un evento asignando el estado de un reloj local al evento inmediatamente después de que ocurra el evento. También se pueden usar números de secuencia simples para este propósito, si la información de tiempo no es importante.

• La *comprobación de estado* verifica la validez o la razonabilidad de operaciones o resultados específicos de un componente. Esta táctica generalmente se basa en el conocimiento del diseño interno, el estado del sistema o la naturaleza de la información bajo escrutinio. Es más a

menudo empleado en interfaces, para examinar un flujo de información específico.

• El *monitoreo de la condición* implica verificar las condiciones en un proceso o dispositivo, o validar las suposiciones hechas durante el diseño. Al monitorear las condiciones, esta táctica evita que un sistema produzca un comportamiento defectuoso. El cálculo de sumas de comprobación es un ejemplo común de esta táctica. Sin embargo, el monitor debe ser simple (e idealmente demostrable) para garantizar que no introduzca nuevos errores de software.

• *Votación*. La realización más común de esta táctica se conoce como triple redundancia modular (TMR), que emplea tres componentes que hacen lo mismo, cada uno de los cuales recibe entradas idénticas, y envía su salida a la lógica de votación, que se utiliza para detectar cualquier inconsistencia entre los Tres estados de salida. Ante una inconsistencia, el elector informa de una falta. También debe decidir qué salida utilizar. Puede dejar que la mayoría gobierne, o elegir un promedio calculado de las salidas dispares. Esta táctica depende fundamentalmente de la lógica de votación, que generalmente se realiza como un singleton simple, rigurosamente revisado y probado, de modo que la probabilidad de error es baja.

• La *replicación* es la forma más simple de votar; Aquí, los componentes son clones exactos el uno del otro. Tener múltiples copias de componentes idénticos puede ser eficaz para proteger contra fallas aleatorias de hardware, pero esto no puede proteger contra errores de diseño o implementación, en hardware o software, porque no hay ninguna forma de diversidad incorporada en esta táctica.

• *La redundancia funcional* es una forma de votación destinada a solucionar el problema de fallas de modo común (fallas de diseño o implementación) en componentes de hardware o software. En este caso, los componentes siempre deben proporcionar la misma salida a la misma entrada, pero están diseñados e implementados de manera diversa.

• *redundancia analítica*permite no solo la diversidad entre las partes privadas de los componentes, sino también la diversidad entre las entradas y salidas de los componentes. Esta táctica tiene la intención de tolerar errores de especificación al usar especificaciones de requisitos separadas. En los sistemas integrados, la redundancia analítica también ayuda cuando es probable que algunas fuentes de entrada no estén disponibles a veces. Por ejemplo, los programas de

aviónica tienen múltiples formas de calcular la altitud de la aeronave, como el uso de la presión barométrica, el altímetro del radar y el uso geométrico de la distancia en línea recta y el ángulo de vista hacia abajo de un punto en el suelo. El mecanismo de votación utilizado con la redundancia analítica debe ser más sofisticado que simplemente permitir que la mayoría gobierne o calcule un promedio simple. Puede que tenga que entender qué sensores son actualmente confiables o no,

• La *detección de excepciones se* refiere a la detección de una condición del sistema que altera el flujo normal de ejecución. La táctica de detección de excepciones se puede refinar aún más:

• *Las excepciones del sistema* variarán de acuerdo con la arquitectura de hardware del procesador empleada e incluirán fallas como la división por cero, fallas de bus y dirección, instrucciones de programas ilegales, etc.

• La táctica de *cercado de parámetros* incorpora un patrón de datos a *priori* (como 0xDEADBEEF) colocado inmediatamente después de cualquier parámetro de longitud variable de un objeto. Esto permite la detección en tiempo de ejecución de sobrescribir la memoria asignada para los parámetros de longitud variable del objeto.

• *La escritura de parámetros* emplea una clase base que define funciones que agregan, encuentran e iteran sobre los parámetros de mensajes con formato de tipo-longitud-valor (TLV). Las clases derivadas utilizan las funciones de clase base para implementar funciones que proporcionan la tipificación de parámetros de acuerdo con la estructura de cada parámetro. El uso de la escritura fuerte para generar y analizar mensajes da como resultado una mayor disponibilidad que las implementaciones que simplemente tratan los mensajes como grupos de bytes. Por supuesto, todo diseño implica compensaciones. Cuando emplea la tipificación fuerte, normalmente intercambia una mayor disponibilidad contra la facilidad de evolución.

• El tiempo de *espera* es una táctica que genera una excepción cuando un componente detecta que éste u otro componente no ha cumplido con sus limitaciones de tiempo. Por ejemplo, un componente que espera una respuesta de otro componente puede generar una excepción si el tiempo de espera excede cierto valor.

• *Autoprueba*. Los componentes (o, más probablemente, los subsistemas completos) pueden ejecutar procedimientos para probar su funcionamiento correcto. Los procedimientos de autoprueba pueden ser iniciados por el propio componente, o invocados de vez en cuando

por un monitor del sistema. Esto puede implicar el empleo de algunas de las técnicas que se encuentran en el monitoreo de condiciones, como las sumas de comprobación.

Recuperarse de las fallas

Las tácticas de recuperación de fallas se refinan en tácticas de *preparación y reparación* y tácticas de *reintroducción* . A estos últimos les preocupa reintroducir un componente fallido (pero rehabilitado) en el funcionamiento normal.

Las tácticas de preparación y reparación se basan en una variedad de combinaciones de reintento de una computación o introducción de redundancia. Incluyen los siguientes:

• *Redundancia activa (repuesto dinámico)*. Esto se refiere a una configuración en la que todos los nodos (activos o de reserva redundantes) en un grupo de protección [2] reciben y procesan entradas idénticas en paralelo, lo que permite que las piezas de reserva redundantes mantengan el estado síncrono con los nodos activos. Debido a que el repuesto redundante posee un estado idéntico al procesador activo, puede asumir el control de un componente fallido en cuestión de milisegundos. El caso simple de un nodo activo y un nodo de repuesto redundante se conoce comúnmente como redundancia 1 + 1 ("uno más uno"). La redundancia activa también se puede usar para la protección de instalaciones, donde los enlaces de red activos y en espera se utilizan para garantizar una conectividad de red de alta disponibilidad.

• *Redundancia pasiva (recambio en caliente)*.Esto se refiere a una configuración en la que solo los miembros activos del grupo de protección procesan el tráfico de entrada; Uno de sus deberes es proporcionar a los repuestos redundantes actualizaciones periódicas del estado. Debido a que el estado mantenido por los repuestos redundantes solo está acoplado de manera flexible con el de los nodos activos en el grupo de protección (siendo la holgura del acoplamiento una función del mecanismo de control empleado entre los nodos activos y redundantes), los nodos redundantes se conocen como repuestos calientes. Dependiendo de los requisitos de disponibilidad del sistema, la redundancia pasiva proporciona una solución que logra un equilibrio entre la táctica de redundancia activa más disponible pero más intensiva en cómputo (y costosa) y la táctica de reserva en frío menos disponible pero significativamente menos compleja (que también es significativamente más económica). (Por unejemplo de

implementación de redundancia pasiva, consulte la sección sobre plantillas de código en el Capítulo 19).

• *Repuesto (repuesto frío)*. La protección en frío se refiere a una configuración en la que los repuestos redundantes de un grupo de protección permanecen fuera de servicio hasta que se produce una conmutación por error, momento en el que se inicia un procedimiento de encendido y reinicio en el repuesto redundante antes de que se ponga en servicio. Debido a su pobre rendimiento de recuperación, la refrigeración en frío es más adecuada para sistemas que solo tienen requisitos de alta confiabilidad (MTBF) en comparación con aquellos que también tienen requisitos de alta disponibilidad.

• *Manejo de excepciones*. Una vez que se ha detectado una excepción, el sistema debe manejarla de alguna manera. Lo más fácil que puede hacer es simplemente bloquearse, pero desde luego es una idea terrible desde el punto de vista de la disponibilidad, la facilidad de uso, la capacidad de prueba y el buen sentido. Hay posibilidades mucho más productivas. El mecanismo empleado para el manejo de excepciones depende en gran medida del entorno de programación empleado, desde códigos de retorno de función simple (códigos de error) hasta el uso de clases de excepción que contienen información útil en la correlación de fallas, como el nombre de la excepción lanzada, el origen de La excepción, y la causa de la excepción lanzada. El software puede usar esta información para enmascarar la falla, generalmente corrigiendo la causa de la excepción y volviendo a intentar la operación.

• *Retroceso*. Esta táctica permite que el sistema vuelva a un estado bueno conocido anterior, conocido como la "línea de retroceso" (retroceso) en el momento de la detección de un fallo. Una vez que se alcanza el buen estado, entonces la ejecución puede continuar. Esta táctica a menudo se combina con tácticas de redundancia activa o pasiva, de modo que después de que se haya producido una reversión, una versión en espera del componente fallido pase a estado activo. La reversión depende de que una copia de un buen estado anterior (un punto de control) esté disponible para los componentes que están retrocediendo. Los puntos de control se pueden almacenar en una ubicación fija y actualizar a intervalos regulares, o en momentos convenientes o significativos en el procesamiento, como al completar una operación compleja.

• *Actualización de software*es otra táctica de preparación y reparación cuyo objetivo es lograr actualizaciones en servicio de imágenes de

código ejecutables de una manera que no afecte al servicio. Esto se puede realizar como un parche de función, un parche de clase o una actualización de software en servicio (ISSU) sin éxito. Un parche de función se usa en la programación de procedimientos y emplea un enlazador / cargador incremental para almacenar una función de software actualizada en un segmento asignado previamente de la memoria de destino. La nueva versión de la función de software empleará los puntos de entrada y salida de la función en desuso. Además, al cargar la nueva función de software, la tabla de símbolos debe actualizarse y la memoria caché de instrucciones debe invalidarse. La táctica de parche de clase es aplicable para objetivos que ejecutan código orientado a objetos, donde las definiciones de clase incluyen un mecanismo de puerta trasera que permite la adición en tiempo de ejecución de los datos y funciones de los miembros.tácticas de redundancia para lograr actualizaciones que no afecten al servicio del software y el esquema asociado. En la práctica, el parche de función y el parche de clase se usan para entregar correcciones de errores, mientras que la actualización de software en servicio sin problemas se usa para entregar nuevas características y capacidades.

• *Reintentar*. La táctica de reintento asume que la falla que causó el fallo es transitoria y el reintento de la operación puede llevar al éxito. Esta táctica se usa en redes y en granjas de servidores donde se esperan fallas y son comunes. Debe haber un límite en el número de reintentos que se intentan antes de que se declare una falla permanente.

• *Ignorar el comportamiento defectuoso*. Esta táctica requiere ignorar los mensajes enviados desde una fuente particular cuando determinamos que esos mensajes son falsos. Por ejemplo, nos gustaría ignorar los mensajes de un componente externo que lanza un ataque de denegación de servicio al establecer filtros de la Lista de control de acceso, por ejemplo.

• La táctica de *degradación* mantiene las funciones más críticas del sistema en presencia de fallas de componentes, eliminando funciones menos críticas. Esto se hace en circunstancias en las que los fallos de componentes individuales reducen con gracia la funcionalidad del sistema en lugar de causar un fallo completo del sistema.

• La *reconfiguración* intenta recuperarse de las fallas de los componentes al reasignar responsabilidades a los recursos

(potencialmente restringidos) que dejaron de funcionar, mientras se mantiene la mayor funcionalidad posible.

La reintroducción es cuando un componente fallado se reintroduce después de que se haya corregido. Las tácticas de reintroducción incluyen lo siguiente:

• La táctica de la *sombra se* refiere a la operación de un componente previamente fallado o en servicio actualizado en un "modo sombra" durante un tiempo predefinido antes de revertir el componente a un rol activo. Durante este tiempo, su comportamiento puede ser monitoreado para ver si es correcto y puede repoblar su estado de manera incremental.

• *resincronización de estados*es un socio de reintroducción de las tácticas de preparación y reparación de redundancia activa y redundancia pasiva. Cuando se usa junto con la táctica de redundancia activa, la resincronización de estado se produce de manera orgánica, ya que los componentes activos y en espera reciben y procesan entradas idénticas en paralelo. En la práctica, los estados de los componentes activos y en espera se comparan periódicamente para garantizar la sincronización. Esta comparación puede basarse en un cálculo de comprobación de redundancia cíclica (suma de comprobación) o, para sistemas que proporcionan servicios críticos para la seguridad, un cálculo de resumen de mensaje (una función hash de una vía). Cuando se usa junto con la táctica de redundancia pasiva (repuesto dinámico), la resincronización de estado se basa únicamente en la información de estado periódico transmitida desde el (los) componente (s) activo (s) al (los) componente (s) en espera (s), generalmente a través del punto de control.

• *reinicio de escalada*es una táctica de reintroducción que permite al sistema recuperarse de fallas al variar la granularidad de los componentes reiniciados y minimizar el nivel de servicio afectado. Por ejemplo, considere un sistema que admite cuatro niveles de reinicio, de la siguiente manera. El nivel más bajo de reinicio (llámelo Nivel 0) y, por lo tanto, que tiene el menor impacto en los servicios, emplea redundancia pasiva (repuesto dinámico), donde se eliminan y recrean todos los subprocesos secundarios del componente defectuoso. De esta manera, solo los datos asociados con los subprocesos secundarios se liberan y reinicializan. El siguiente nivel de reinicio (Nivel 1) libera y reinicializa toda la memoria desprotegida (la memoria protegida permanecería intacta). El siguiente nivel de reinicio (Nivel 2) libera y

reinicializa toda la memoria, tanto protegida como desprotegida, lo que obliga a todas las aplicaciones a recargarse y reinicializarse. Y el nivel final de reinicio (Nivel 3) implicaría volver a cargar completamente y reinicializar la imagen ejecutable y los segmentos de datos asociados. El soporte para la táctica de reinicio escalado es particularmente útil para el concepto de degradación elegante, donde un sistema puede degradar los servicios que proporciona al tiempo que mantiene el soporte para aplicaciones de misión crítica o de seguridad.

• El *reenvío directo* (NSF) es un concepto que se originó en el diseño del enrutador. En este diseño, la funcionalidad se divide en dos partes: supervisión, o plano de control (que administra la conectividad y la información de enrutamiento), y plano de datos (que realiza el trabajo real de enrutar paquetes desde el remitente hasta el receptor). Si un enrutador experimenta la falla de un supervisor activo, puede continuar enviando paquetes a lo largo de rutas conocidas, con enrutadores vecinos, mientras se recupera y valida la información del protocolo de enrutamiento. Cuando se reinicia el plano de control, implementa lo que a veces se denomina “reinicio correcto”, reconstruyendo de forma incremental su base de datos de protocolo de enrutamiento incluso cuando el plano de datos continúa operando.

Prevenir las faltas

En lugar de detectar fallas y luego tratar de recuperarse de ellas, ¿qué pasaría si su sistema pudiera evitarlas en primer lugar? Aunque esto suena como una cierta medida de clarividencia podría ser necesaria, resulta que en muchos casos es posible hacer eso. [3]

• *Retiro del servicio*. Esta táctica se refiere a colocar temporalmente un componente del sistema en un estado fuera de servicio con el fin de mitigar posibles fallas del sistema. Un ejemplo consiste en poner fuera de servicio un componente de un sistema y reiniciarlo para eliminar fallas latentes (por ejemplo,como fugas de memoria, fragmentación o errores de software en un caché no protegido) antes de que la acumulación de fallas afecte el servicio (lo que resulta en una falla del sistema). Otro término para esta táctica es *el rejuvenecimiento del software* .

• *Transacciones* . Los sistemas dirigidos a servicios de alta disponibilidad aprovechan la semántica transaccional para garantizar que los mensajes asíncronos intercambiados entre componentes distribuidos sean *atómicos* , *coherentes* , *aislados* y *duraderos* . Estas cuatro propiedades se denominan "propiedades ACID". La realización

más común de la táctica de transacciones es el protocolo de "compromiso en dos fases" (también conocido como 2PC). Esta táctica evita las condiciones de carrera causadas por dos procesos que intentan actualizar el mismo elemento de datos.

• *Modelo predictivo* . Un modelo predictivo, cuando se combina con un monitor, se emplea para monitorear el estado de salud de un proceso del sistema para asegurar que el sistema está operando dentro de sus parámetros operativos nominales, y para tomar medidas correctivas cuando se detectan condiciones que predicen un futuro probable. faltas Las métricas de rendimiento operacional monitoreadas se utilizan para predecir la aparición de fallas; los ejemplos incluyen la tasa de establecimiento de sesión (en un servidor HTTP), el cruce de umbrales (monitoreo de marcas de agua altas y bajas para algunos recursos compartidos restringidos), o el mantenimiento de estadísticas para el estado del proceso (en servicio, fuera de servicio, en mantenimiento, inactivo), mensaje estadísticas de longitud de cola, y así sucesivamente.

• *Prevención de excepciones*. Esta táctica se refiere a las técnicas empleadas con el fin de evitar que ocurran excepciones al sistema. El uso de clases de excepción, que permite que un sistema se recupere de forma transparente de las excepciones del sistema, se trató anteriormente. Otros ejemplos de prevención de excepciones incluyen tipos de datos abstractos, como punteros inteligentes y el uso de envoltorios para evitar fallas, como punteros colgantes y violaciones de acceso a semáforos. Los punteros inteligentes previenen las excepciones al realizar la comprobación de los límites de los punteros y al garantizar que los recursos se desasignen automáticamente cuando no se hace referencia a los datos. De esta manera se evitan las fugas de recursos.

• *Aumentar el conjunto de competencias*.El conjunto de competencias de un programa es el conjunto de estados en los que es "competente" para operar. Por ejemplo, el estado cuando el denominador es cero está fuera del conjunto de competencias de la mayoría de los programas de división. Cuando un componente genera una excepción, está indicando que se ha descubierto que está fuera de su conjunto de competencias; En esencia, no sabe qué hacer y está tirando la toalla. Aumentar el conjunto de competencias de un componente significa diseñarlo para manejar más casos (fallas) como parte de su operación normal. Por ejemplo, un componente que asume que tiene acceso a un recurso compartido puede generar una excepción si

descubre que el acceso está bloqueado. Otro componente puede simplemente esperar el acceso o regresar inmediatamente con una indicación de que completará su operación la próxima vez que tenga acceso. En este ejemplo,

5.3. UNA LISTA DE VERIFICACIÓN DE DISEÑO PARA DISPONIBILIDAD

La Tabla 5.4 es una lista de verificación para respaldar el proceso de diseño y análisis de disponibilidad.

Tabla 5.4. Lista de verificación para apoyar el proceso de diseño y análisis de disponibilidad

Category	Checklist
Allocation of Responsibilities	Determine the system responsibilities that need to be highly available. Within those responsibilities, ensure that additional responsibilities have been allocated to detect an omission, crash, incorrect timing, or incorrect response. Additionally, ensure that there are responsibilities to do the following: ▪ Log the fault ▪ Notify appropriate entities (people or systems) ▪ Disable the source of events causing the fault ▪ Be temporarily unavailable ▪ Fix or mask the fault/failure ▪ Operate in a degraded mode
Coordination Model	Determine the system responsibilities that need to be highly available. With respect to those responsibilities, do the following: ▪ Ensure that coordination mechanisms can detect an omission, crash, incorrect timing, or incorrect response. Consider, for example, whether guaranteed delivery is necessary. Will the coordination work under conditions of 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 used (processors, communications channels, persistent storage, and processes). For example, 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. For example, how much lost information can the coordination model withstand and with what consequences?
Data Model	Determine which portions of the system need to be highly available. Within those portions, determine which data abstractions, along with their operations or their properties, 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. For example, ensure that write requests are cached if a server is temporarily unavailable and performed when the server is returned to service.
Mapping among Architectural Elements	Determine which artifacts (processors, communication channels, persistent storage, or processes) may produce a fault: omission, crash, incorrect timing, or incorrect response. Ensure that the mapping (or remapping) of architectural elements is flexible enough to permit the recovery from the fault. This may involve a consideration of the following: ▪ Which processes on failed processors need to be reassigned at runtime ▪ Which processors, data stores, or communication channels can be activated or reassigned at runtime ▪ How data on failed processors or storage can be served by replacement units

	▪ How quickly the system can be reinstalled 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. For example, it is possible to write one module that contains code appropriate for both the active component and backup components in a protection group.
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 the source of events causing the fault; be temporarily unavailable; 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.

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. For example: ▪ If late binding is used to switch between artifacts such as processors that will receive or be the subject of faults, will the chosen 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 milliseconds, 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?
Choice of Technology	Determine the available technologies that can (help) detect faults, recover from faults, or reintroduce 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?

5.4. RESUMEN

La disponibilidad se refiere a la capacidad del sistema para estar disponible para su uso, especialmente después de que se produce una falla. La falla debe ser reconocida (o prevenida) y luego el sistema debe responder de alguna manera. La respuesta deseada dependerá de la criticidad de la aplicación y del tipo de falla y puede ir desde "ignorarla" a "seguir adelante como si no hubiera ocurrido".

Las tácticas de disponibilidad se clasifican en detectar fallas, recuperarse de fallas y prevenir fallas. Las tácticas de detección dependen, esencialmente, de la detección de signos de vida de varios componentes. Las tácticas de recuperación son una combinación de reintentar una operación o mantener datos o cálculos redundantes. Las tácticas de prevención dependen de la eliminación de elementos del servicio o de la utilización de mecanismos para limitar el alcance de las fallas.

Todas las tácticas de disponibilidad involucran el modelo de coordinación porque el modelo de coordinación debe ser consciente de las fallas que ocurren para generar una respuesta apropiada.