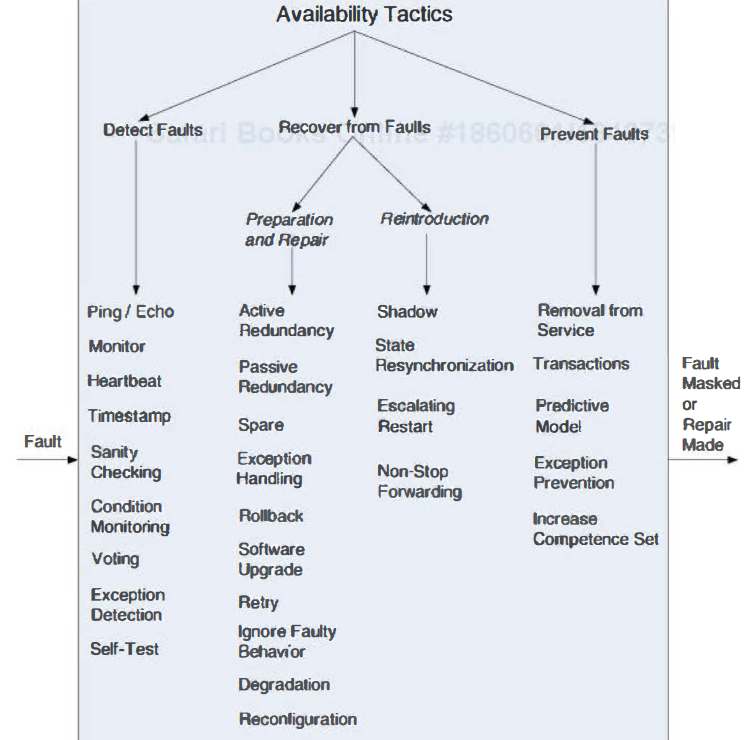
Libro Bass Kazman-Software Architecture in Practice 2013

Availability página 106

El escenario

|  |  |
| --- | --- |
| Fuente de estimulo | Hay 2 tipos fuente: interna y externa del origen de falla, esto depende del tipo de sistema que se analiza. |
| Estimulo | La falla puede ser :  Omisión: el componente falla en responder a una entrada (input).  Crash: el componente repetidamente sufre faults por omisión.  Timing: el componente responde pero la respuesta está fuera de tiempo (demasiado temprano o demasiado tarde).  Response: un componente responde con un valor incorrecto. |
| Artefactos | Se especifica que el recurso que el recurso que es requerido es:  Altamente disponible como procesador, canal de comunicación, proceso o almacenamiento. |
| Entorno | El estado del sistema cuando la fault o failure ocurre y tambien puede afectar la respuesta deseada del sistema. Ejemplo: el sistema ha tenido fallas y esta operando en modo no normal, que hace factible apagar el sistema (shut down), si es la primera falla, puede elegir trabajar con el sistema a menor capacidad (degradado). |
| Respuesta | Hay un numero de opciones ante el fault de un sistema  la falla debe ser detectada y aislada antes de otras respuestas  la falla debe ser detectada y corregirse, las soluciones pueden ser guardado en  bitácora, notificar usuarios, limitar o mitigar el daño del fault,  cambiar a modo degradado que puede ser con menos funcionalidad o capacidad  apagar el sistema (shutdown) o deshabilitar el sistema |
| Medición de la respuesta | La respuesta puede ser  un porcentaje de disponibilidad (availability)  especificar un tiempo para detectar la falla  tiempo para reparar el fault  tiempo o intervalos de tiempo donde el sistema deba estar disponible  tiempo de espera en que el sistema va a estar disponible. |

Availability es la propiedad de software de que esté presente y listo para realizar una tarea cuando sea necesario, es un concepto amplio y que se puede traslapar con disponibilidad y confiabilidad.



**1 Detectar fallas**

1.1 Ping / echo : es el test de una solicitud o respuesta asíncrona entre nodos, se usa para determinar el alcance y conexión de una red, se verifica que un componente responde, es decir “esta vivo”, indica el tiempo de recorrido de un mensaje, existen los programas que verifican la disponibilidad de los nodos.

1.2 Monitor: es un componente que es para monitorear el estado de salud de varias partes del Sistema como: cpu, procesos, entradas y salidas, memoria; puede detector fallas o congestiones en la red, puede iniciar auto-tests, o detector pequeñas anomalías.

1.3 Heartbeat es un mecanismo de detección de falla que usa periódicamente un mensaje a un sistema de monitoreo y verificación de un proceso, un ejemplo es cuando el proceso a ser monitoreado resetea el watchdog timer lo cual lo priva de expirar y genera un fault.

Para sistemas donde la escalabilidad es una prioridad y puede haber sobrecarga por estos reseteos.

La diferencia entre heartbeat y ping/echo es quien tiene la responsabilidad de iniciar el health check, el monitor o el componente mismo.

1.4 time stamps: esta táctica se usa para detector secuencias de eventos incorrectas, primariamente sistemas de envío de mensajes distribuidos, el *time stamps* de un evento puede ser fijado al asignar el estado de un clock local al evento inmediatamente después de que el evento ocurra, secuencias simples de números pueden ser usados.

1.5 sanity checking checa la validez de lógica de operaciones especificas o salidas de un componente. La táctica es típicamente basada en el conocimiento del diseño interno, el estado del sistema, o la naturaleza de la información, se usa más en las interfaces para examinar el flujo de información.

1.6 Condition monitoring: consiste en checar las condiciones en un proceso o dispositivo, o validación de asunciones (lo que se asume) hechas durante el diseño.

Al monitorear las condiciones se prevé una falla, un ejemplo es el cálculo de checksums para asegurar que no se introduzcan errores.

1.7 Voting : la realización más común de esta táctica se refiere a la triple modular redundancy

(TMR), que emplea 3 componentes que hagan la misma tarea, cada componente recibe el mismo input, se revisan las salidas de los componentes para detectar alguna diferencia.

Tipos de voting

1.7.1 Replication, es el voting más simple, los componentes son copias idénticas y es más

sencillo determinar diferencias.

1.7.2 functional redundancy es una forma de voting para determinar la causa de common-

mode failures, (fallas de diseño o implementación) en hardware o componentes de

software, la prueba consiste en la consistencia de resultados entre componentes

1.7.3 Analytic redundancy: se pueden manejar componentes diferentes, el análisis es

diferente porque los outputs muchas veces son diferentes, el resultado inicial es

cuales son los componentes o sensors que están disponibles

1.8 detección de excepciones: se refiere a la detección de una condición del sistema que altera el flujo normal de ejecucion.

1.8.1 System exceptions: esta categoría variara dependiendo del procesador, de la

arquitectura, las fallas como división por cero, fallas de bus y direcciones de

memoria, etc.

1.8.2 parameter fence : son tácticas que incorporan patrones de datos “a priori”

insertadas inmediatamente después de alguna variable de longitud de un objeto,

esto permite la detección de sobre-escritura de memoria localizada para las variables

de longitud.

1.8.3 Parameter typing emplea una clase base que define funciones como agregar,

encontrar sobre TLV type-length-value formatted message parameters. Las clases

derivadas proveen parameter typing de acuerdo a cada estructura de parametros, el

uso de strong typing para construir o parsear mensajes resultan en mayor

availability, aunque el diseño involucra tradeoffs.

Cuando usas strong typing estas compensando la alta availability contra la facilidad

de evolución.

1.8.4 Timeout: la táctica que genera una excepción cuando un componente detecta

cuando cualquier componente que falla para satisfacer las restricciones de tiempo.

Ejemplo: un componente que espera respuesta de otro componente que excedió el

tiempo para contestar.

1.9 Self-test es cuando componentes (o más probable un subsistema) que puede correr

funciones para probarse a sí mismos, puede incluir algunas técnicas como monitoring

o checksums.

**2 Recover from faults**

Estas tácticas si son refinadas pueden llegar a ser tácticas “preparación y reparación” y tácticas de “re-introducción” que es cuando reintroducen un componente que había fallado y que ahora funciona bien.

**2.1 tácticas de “preparación y reparación”**

2.1.1 Active redundancy (hot spare): refiere a la configuración donde todos los nodos (activos o redundant spare) en un grupo de prueba (grupo de protección) donde un grupo recibe y procesa inputs idénticos en paralelo permitiendo el redundant spare para mantener un estado síncrono con nodos activos. La active redundancy puede ser usada acciones de protección donde una red activa y disponible se enlazan para asegurar una alta disponibilidad.

Grupo de protección es un grupo de nodos procesadores donde un nodo o más, están activos con los demás nodos sirviendo como redundant spares.

2.1.2 passive redundancy (warm spare): se refiere a una configuración donde el único nodo active está procesando entradas de tráfico, su objetivo es crear redundant spare con updates periódicos, los nodos redundantes se conoce como warm spare, dependiendo de la disponibilidad de los recursos del sistema, la redundancia pasiva provee una solución para alcanzar el balance entre mayor disponibilidad con mayor poder de computo (que es caro), la tactica cold spare es más barata que la táctica active redundancy.

2.1.3 spare (cold spare) cold sparing se refiere a la configuración donde el spare redundant de un grupo de protección permanece fuera de servicio hasta que ocurra un fail-over, debido a su pobre rendimiento de recuperación, cold sparing aplica mejor para sistemas con alta reliability de requerimientos contrario a aquellos que tengan alta availability de requerimientos.

2.1.4 exception handling: una vez que los requerimientos ha sido detectada el Sistema tiene que manejarla de algún modo, el manejo de la excepción depende del lenguaje (mensaje de error, etc), en general solo se enmascara la falla y se corrige la causa de la excepción y recuperar la operación al intentarla otra vez.

2.1.5 Rollback: esta táctica permite que el sistema revierta un estado previo bueno, referido como “rollback line”, después de la detección de la falla, una vez que el buen estado es alcanzado la ejecución puede terminar.

2.1.6 software upgrade: alcanzar objetivos mediante ejecuciones que no afecten la globalidad del software en el formato hitless in-service que agrega nuevas funcionalidades y capacidades.

2.1.7 Retry: esta táctica asume que el fault que causa la failure es temporal y reiterando la operación puede llevar a la operación exitosa (ejemplo: salga y vuelva a entrar), se usa en redes donde las fallas son comunes, debería considerarse un numero de máximo de retry (intentos) antes de declarar como failure permanente.

2.1.8 Ignore faulty behaviour: esta táctica ignora mensajes enviados desde una fuente particular donde se considera que los mensajes no tienen valor ni validez.

2.1.9 degradation : esta táctica mantiene las funciones más críticas del sistema en la presencia de failures, suspendiendo algunas funciones críticas, esto aplica en sistemas donde la suspensión de algunas funciones no afecta al sistema completo.

2.1.10 reconfiguration: se intenta recuperar el sistema desde fallas en componentes al reasignar responsabilidades a los componentes funcionando, en la medida de lo posible para mantener la funcionalidad.

**2.2 Re-introducción**

Es cuando se reintroducen componentes que fallaron y se ha corregido

2.2.1 shadow: táctica que introduce a operación un in-service upgraded component en modo “shadow mode” por un tiempo determinado antes de pasarlo a un rol activo.

Durante este periodo la conducta puede ser monitoreada para su corrección.

2.2.2 State resynchronization: es la reintroducción de un socio para la redundancia activa y redundancia pasiva, puede ser un proceso sin mucho tramite porque se revisa periódicamente el cyclic redundancy check calculation (checksum).

2.2.3 escalating restart: permite al sistema recobrarse de los faults al variar la granularidad de los componentes recomenzados y minimizando el nivel de servicio.

2.2.4 Non-stop forwarding (NSF) es un concepto que nace en el diseño de routers, donde el diseño de la funcionalidad se divide en 2 partes: no. **1 supervisor o control plano** que maneja conectividad e información de router, no. **2 data plane** que hace el trabajo de rutear paquetes del emisor al receptor.

**3 Prevent Faults**

Qué tal si en vez de detectar fallas y corregirlas el sistema es capaz de prevenirlas, aunque parezca un ejercicio de clarividencia puede funcionar.

La mejor manera es crear código de programación de calidad con inspecciones de código, programación por pares, revisiones de requerimientos efectivas y cualquier buena práctica de ingeniería de software.

3.1 removal from service: la táctica refiere a poner un componente temporalmente como fuera de servicio para mitigar los efectos mientras se corrigen sus errores, también se le conoce como *software rejuvenation*.

3.2 transactions: asegurar que la parte del sistema con mayor tráfico se verifique que los mensajes asíncronos que se intercambian los componentes distribuidos sean : atómicos. Consistentes, aislados y durables. Estas propiedades se llaman “propiedades ACID” (en inglés)

3.3 predictive model: cuando se combina un modelo predictivo con un sistema de monitoreo para verificar el estado de salud del sistema, que significa que opere con parámetros saludables de funcionamiento. Ejemplos: tasas de rendimiento de servidores HTTP, establecimiento de límites o estadísticas para uso de recursos.

3.4 Increase competence set: se define un conjunto de estados en donde es “competente” operar, dicho de otra manera, incrementar la competencia de un componente es diseñar más casos de posibles fallas que son parte de la operación normal, ejemplo: revisar más casos de excepción.

**Actividad**: hacer una lista de las tácticas que no pueden faltar y las que se pueden omitir, en orden de prioridad, un análisis de causas y efecto como lo es la ley de Pareto para el sistema a implementar.

De la lista de tácticas seleccionadas contrastar contra ……

**Checklist de diseño (contrastar con Categorías de decisiones de diseño arquitecturales)**

**Son para contrastar si las decisiones tomadas cumplen esas preguntas o sentencias (asignaciones)**

|  |  |
| --- | --- |
| Asignación de responsabilidades | Adicionalmente debe asignar  -bitácora de falla  -notificar a las personas o sistemas adecuadas  -deshabilitar la fuente de faults  -no estar disponible (temporalmente)  -corregir o enmascarar la fault o failure  -operar en modo degradado |

|  |  |
| --- | --- |
| Modelo de coordinación | Definir responsabilidades que necesitan estar obligatoriamente disponibles  -asegurar que los mecanismos de coordinación detecten omisiones, crash, timing incorrecto, respuestas incorrectas.  La coordinación puede trabajar en modo degradado ?  -asegurar que los mecanismos de coordinación hagan: bitacora de falla, -notificar a las personas o sistemas adecuadas, deshabilitar la fuente de faults, -no estar disponible (temporalmente), corregir o enmascarar la fault o failure, operar en modo degradado  -asegurar que los mecanismos de coordinación soporten el reemplazo de artefactos usados (procesadores, canales de comunicación, almacenamiento, procesos) para que siga funcionando el sistema.  La coordinación puede trabajar en modo degradado ? |

|  |  |
| --- | --- |
| Modelo de datos | Determinar que porciones del sistema necesita obligatoriamente estar disponible.  -Que datos, abstracciones de datos, además de sus operaciones, propiedades que puedan causar faults, omisiones, crash, timing incorrecto, respuestas incorrectas.  -asegurar que datos, abstracciones de datos, además de sus operaciones, propiedades puedan ser deshabilitados, deshabilitados temporalmente, corregidos o enmascarados en el evento de un fault. |

|  |  |
| --- | --- |
| Mapeo entre elementos arquitecturales | Determinar que artefactos (procesadores, canales de comunicación, almacenamiento, procesos) puedan causar faults, omisiones, crash, timing incorrecto, respuestas incorrectas.  Asegurar que el mapeo o remapeo de elementos arquitecturales es flexible suficiente para recuperarse del fault. Las preguntas serian:  -cual proceso o procesador necesita ser reasignado en tiempo de ejecución?  -que procesador, almacenamiento, canales de comunicación pueden ser activados o reasignados en tiempo de ejecución.  -como los datos de un procesador o almacenamiento que fallo puede ser conectado por las unidades reemplazantes.  -que tan rápido el sistema puede ser reinstalado  -como reasignar elementos como procesadores, canales de comunicación y almacenamiento.  -se usan tácticas que manejen la redundancia de funcionalidad, el mapeo de modulos a componentes redundantes es importante. |

|  |  |
| --- | --- |
| Administración de recursos | Determinar que recursos críticos son necesarios para que el sistema siga funcionando cuando haya un fault, omisiones, crash, timing incorrecto, respuestas incorrectas.  -asegurar que haya los suficientes recursos en stock para cualquier evento |

|  |  |
| --- | --- |
| Binding time | -Determinar cómo y cuándo los elementos arquitecturales estarán en interacción,  -Si hay interacción tardía entre elementos, los elementos a interactuar podrán realizar la interacción ?  -Los tiempos de tolerancia son reducido o excedidos de manera que puedan causar un fault ? |

|  |  |
| --- | --- |
| Selección de tecnología | -Determinar que la tecnología disponible pueda o ayude a detectar faults, recuperarse de faults, o reintroducir componentes que han fallado.  -Determinar que tecnologías disponibles ayudarían a responder ante un fault  -determinar la disponibilidad de las tecnologías disponibles.  Que fallas se pueden recuperar ?  Que fallas pueden introducir la tecnología seleccionada ? |