



SISTEMAS DISTRIBUTIVOS

TÉCNICAS DE TOLERANCIA Y RECUPERACIÓN

TÉCNICAS DE TOLERANCIA Y RECUPERACIÓN


En los sistemas distribuidos, donde múltiples nodos interactúan a través de redes propensas a fallos, la capacidad de tolerar y recuperarse de errores no es una opción, sino un requisito fundamental. A diferencia de los sistemas centralizados, en los que la interrupción de un componente puede ser fácilmente contenida, un fallo en un sistema distribuido puede propagarse, comprometer el rendimiento global o incluso derivar en pérdida de datos. Por ello, se aplican técnicas especializadas de tolerancia y recuperación, orientadas a mantener la disponibilidad del servicio y la integridad de la información ante eventos inesperados (Costas Santos, 2015).

Este subtema explora dichas técnicas, organizadas, según su propósito: evitar que un fallo impacte al sistema (tolerancia) y restaurar el funcionamiento una vez que el fallo ha ocurrido (recuperación).


1. Tolerancia a fallos: diseño para la continuidad.

La tolerancia a fallos se refiere a la capacidad del sistema para seguir operando correctamente incluso cuando uno o más componentes fallan. Esta propiedad se logra mediante la implementación de técnicas que previenen la interrupción del servicio frente a errores previsibles (Costas Santos, 2015).

- a. **Redundancia.** La estrategia más clásica consiste en duplicar componentes claves (hardware, software, datos) para asegurar que si uno falla, otro pueda continuar su función.

 **Ejemplo.** En un clúster web, múltiples servidores sirven la misma aplicación; si uno cae, un balanceador de carga redirige el tráfico a los demás.

- b. **Replicación de datos.** La replicación permite mantener copias sincronizadas de datos en distintos nodos, garantizando disponibilidad y consistencia parcial incluso ante fallos locales.


 **Ejemplo.** En sistemas como Cassandra o MongoDB, cada fragmento de datos se replica en al menos tres nodos para tolerancia geográfica y de hardware.

- c. **Tolerancia mediante quorum.** Los sistemas que implementan consenso mediante quorum (como Raft o Paxos) permiten que el sistema continúe operando si una mayoría de nodos está activa, sin necesidad de que todos estén disponibles (Urbano López, 2015).


2. Técnicas de recuperación: restaurar la operación.

La recuperación busca restablecer el estado del sistema una vez que se ha producido una falla, sin pérdida de integridad ni continuidad lógica.

- a. **Checkpoints (puntos de control).** Consiste en guardar el estado actual del sistema de manera periódica. Si ocurre una falla, el sistema puede restaurarse desde el último checkpoint sin reiniciar desde cero.

 **Ejemplo.** En cálculos científicos distribuidos, se guarda el estado de los procesos cada 10 minutos para evitar repetir todo el trabajo en caso de caída.

- b. **Logs de transacciones.** Los sistemas de bases de datos distribuidas suelen mantener un log o bitácora de todas las operaciones. Si ocurre un fallo, se rehacen las operaciones registradas hasta el último punto coherente (Urbano López, 2015).


 **Ejemplo.** PostgreSQL utiliza Write-Ahead Logging (WAL), que permite reconstruir transacciones pendientes luego de un reinicio inesperado.

- c. **Reconexión automática y reintentos.** Muchos servicios implementan estrategias de reintento automático ante fallos de red o de nodo, con reconfiguración dinámica del clúster para redistribuir tareas o líderes.

 **Ejemplo.** En Apache Kafka, si un broker cae, el controlador reasigna sus particiones a otros brokers sin intervención humana.

3. Combinación de estrategias para mayor resiliencia.

Las técnicas de tolerancia y recuperación no deben verse como soluciones independientes, sino como componentes complementarios de una estrategia robusta. Un sistema verdaderamente confiable no solo previene caídas, sino que también se recupera rápida y ordenadamente cuando estas son inevitables (Urbano López, 2015).

 **Ejemplo combinado.** Un sistema de pagos electrónicos puede utilizar replicación para mantener la disponibilidad de sus datos, consenso con quorum para validar operaciones, y logs de transacciones para recuperarse de caídas de servidores.

4. Consideraciones de diseño.




-  **Balance entre costos y seguridad:** más replicación y checkpoints implican mayor consumo de recursos.
-  **Impacto sobre la latencia:** algunas técnicas (como los logs o el quorum) agregan retardo al sistema.
-  **Escenarios específicos:** no todas las técnicas son ideales para todos los entornos. Por ejemplo, los sistemas con baja tolerancia a pérdida de datos deben priorizar la persistencia inmediata.

Figura 1. Técnicas de replicación

