Replicación, Consistencia y Tolerancia a Fallas

Replicación, Consistencia y Tolerancia a Fallas

- La replicación juega un importante rol en los sistemas distribuidos y, por lo general, se utiliza para incrementar la confiabilidad y mejorar el rendimiento de un sistema, lo que implica garantizar que todas las copias del sistema sean actualizadas.
- Un reto importante en la consistencia es con respecto a los datos compartidos, que son accedidos por varios procesos simultáneamente, ya que implementar la consistencia crece en complejidad conforme el sistema distribuido es escalado

Replicación, Consistencia y Tolerancia a Fallas

Sobre la consistencia.

- La administración de la réplica, donde se consideran aspectos como la ubicación de los servidores de réplicas y distribución del contenido entre estos servidores.
- El mantenimiento de consistencia de las réplicas, lo cual implica que las actualizaciones de todas las réplicas se deben de realizar de una manera rápida (casi inmediata).
- Por otro lado, también los sistemas distribuidos son diseñados de manera que puedan recuperarse automáticamente de fallas parciales, sin que se afecte seriamente el desempeño total.

Replicación, Consistencia y Tolerancia a Fallas

- Una falla se puede presentar en un sistema distribuido y afectar el funcionamiento de algunos componentes, sin embargo, el sistema puede seguir operando. Esto no es posible en un sistema no distribuido.

Replicación

- La replicación significa mantener copias de una información en múltiples computadoras. Este recurso es ampliamente usado en los sistemas distribuidos, debido a que proporciona un mejor rendimiento, alta disponibilidad y tolerancia a fallas. Algunos ejemplos donde se usa la replicación son [Coulouris *et al*., 2012]:
  - Almacenamiento en caché en los servidores web y servidores proxy web.
  - El servicio de nombres DNS.

Beneficios de usar replicación en los sistemas distribuidos

Mejoras del rendimiento

Entre las características con respecto al rendimiento del sistema distribuido que deben de ser observadas al replicar datos se pueden mencionar las siguientes [Coulouris et al., 2012]:

- La replicación se implementa principalmente a través de cachés en clientes o servidores.
- Es importante mantener copias de los resultados obtenidos en llamadas anteriores al servicio para evitar o reducir el coste de llamadas idénticas.
- Se evita el tiempo de latencia derivado del cálculo del resultado o de realizar consultas a otros servidores.
- La replicación de datos con actualizaciones muy frecuentes en la red genera un costo en el uso de protocolos de intercambio y actualización, además de limitar la efectividad de la réplica.

Beneficios de usar replicación en los sistemas distribuidos

Alta disponibilidad

Entre las características con respecto a la disponibilidad del sistema distribuido que deben de ser observadas al replicar datos se pueden mencionar las siguientes [Coulouris et al., 2012]:

- La proporción de tiempo que un servicio está accesible con tiempos de respuesta razonables, que debe de ser cercana al 100%.

Existen pérdidas de disponibilidad debido a los siguientes factores:

- Fallas en el servidor. Si se replica n veces el servidor, la disponibilidad será $1-p^n$. Por ejemplo, para n = 2 servidores, la disponibilidad es $1 - 0.05^2 = 1 – 0.0025 = 99.75\%$.
- Particiones de red o desconexiones.
- Operación desconectada (son aquellas desconexiones de comunicación que a menudo no se planifican y son un efecto secundario de la movilidad del usuario).

Beneficios de usar replicación en los sistemas distribuidos

Tolerancia a fallas

Entre las características respecto a la tolerancia a falla en los sistemas distribuidos, se pueden mencionar las siguientes [Coulouris et al., 2012]:

- Una alta disponibilidad no implica necesariamente corrección, esto se debe a que puede haber datos no actualizados, o inconsistentes, originados por procesos concurrentes.
- Se puede utilizar la replicación para alcanzar una mayor tolerancia a fallas.
- Ante una caída o suspensión de servidores; por ejemplo, si se tienen n servidores, pueden fallar n-1 sin que el servicio sufra alteración.
- Ante fallas bizantinas, es decir, si f servidores presentan fallas bizantinas, un sistema con 2f+1 servidores proveería un servicio correcto.

Requisitos para realizar la replicación

Entre los requisitos más importantes que se deben considerar al implementar la replicación en un sistema distribuido, están:

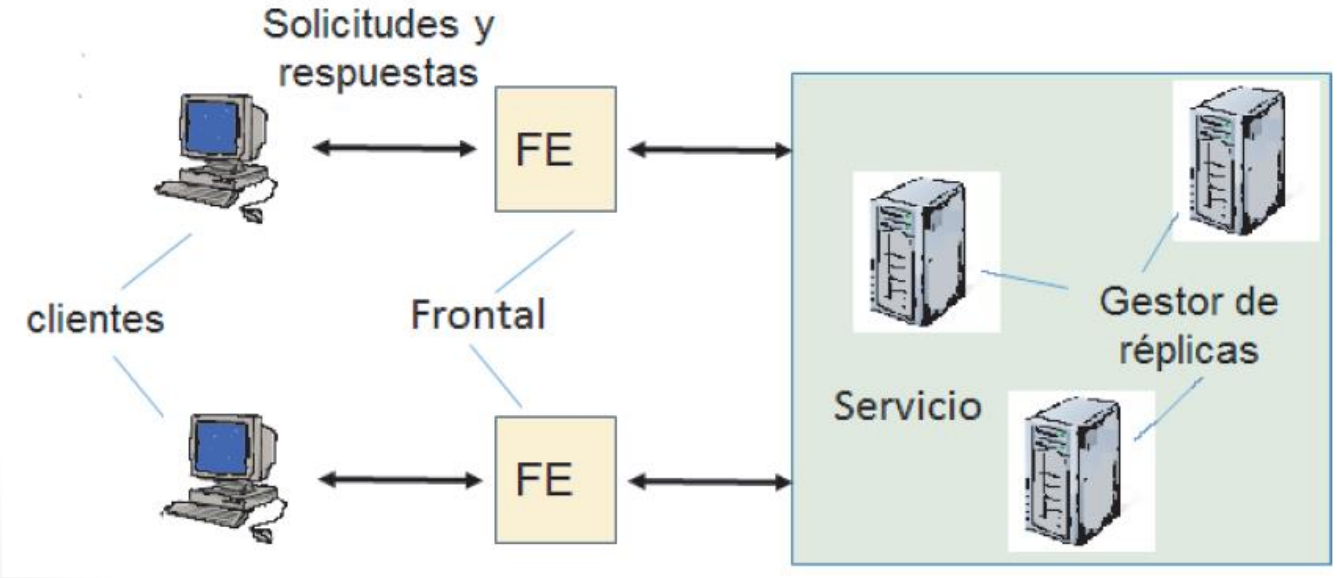
- Transparencia.
- Consistencia.

Requisitos para realizar la replicación

- El requisito de transparencia implica que los clientes no son conscientes de que hay múltiples copias del recurso al que ellos están accediendo. Para los usuarios, solo existen recursos lógicos individuales.
- La consistencia implica que las operaciones sobre un conjunto de objetos replicados deben de dar resultados que sigan la especificación de corrección definida para esos objetos. De acuerdo con el tipo de aplicación, la consistencia puede ser más o menos estricta.

Modelo general de gestión de réplica

- Esta sección presenta un modelo de sistema general para la gestión de réplicas. Además, describe el papel de los sistemas de comunicación grupal en la consecución de la tolerancia a fallas a través de la replicación. El modelo general de gestión de réplica



Modelo general de gestión de réplica

- Los componentes principales de esta arquitectura de gestión de datos replicados son:
  - Gestor de réplicas.
  - Frontal (front-end).
- El gestor de réplicas son los componentes que almacenan las réplicas de un determinado objeto o servicio y operan sobre ellas. Frontal (FE) es el componente o interfaz que atiende las llamadas de los clientes y se comunica con los gestores de réplicas.

Modelo general de gestión de réplica

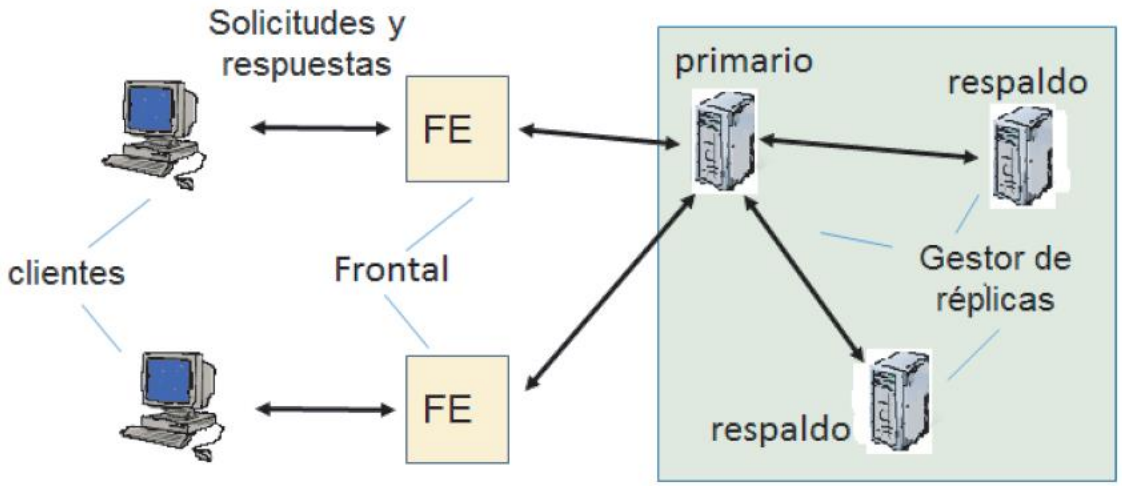
Cinco fases de una única solicitud a los objetos replicados.

1. La petición es enviada por el frontal a uno o más gestores con las siguientes opciones:
   - Se envía la petición a un gestor y este la reenvía a otros.
   - Multidifundir la petición a varios gestores.
2. Los gestores se coordinan para ejecutar la petición de manera consistente. Los tipos de ordenación para que los gestores manipulen las réplicas son:
   - Ordenamiento FIFO.
   - Ordenamiento casual.
   - Ordenamiento total.
3. Se ejecuta la petición, la cual podría ser realizada de manera tentativa.
4. Se llega a un acuerdo o consenso antes de consumar la ejecución.
5. Uno o más gestores de réplicas entrega una respuesta al frontal.

Servicios de tolerancia a fallas basados en replicación

Replicación pasiva

- En la replicación pasiva existe un gestor de réplicas primario y uno o más gestores secundarios también conocidos como “respaldos” o “esclavos”. Los frontales se comunican con el gestor primario, quien ejecuta las operaciones y manda copias a los respaldos. Si el gestor de réplicas primario falla, entonces un gestor de réplicas de respaldo lo sustituye



Servicios de tolerancia a fallas basados en replicación

Replicación pasiva

- Una replicación pasiva tiene las siguientes fases de ejecución
  - Petición.
  - Coordinación.
  - Ejecución.
  - Acuerdo.
  - Respuesta.

Servicios de tolerancia a fallas basados en replicación

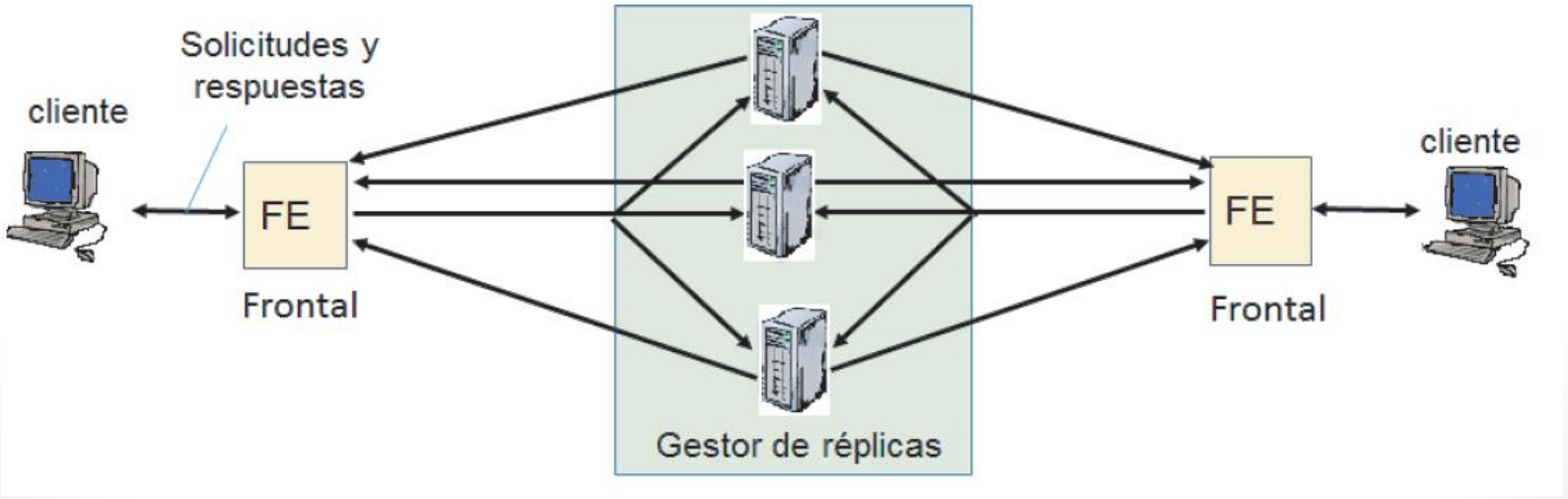
Replicación pasiva

- Durante la fase de petición, el frontal envía la petición al gestor primario, quien activa la fase de coordinación y ejecuta las peticiones siguiendo una ordenación FIFO. Se realiza la fase de ejecución de la petición y se almacena la respuesta. Si es una petición de actualización, entonces en la fase de acuerdo el gestor primario envía la actualización a todos los respaldos, quienes confirman la recepción.
- Finalmente, en la fase de respuesta, el gestor primario envía la respuesta correspondiente al frontal. La replicación pasiva tolera fallas de procesos pero no tolera fallas bizantinas. El frontal requiere poca funcionalidad, sin embargo, este esquema de replicación es propenso a problemas de cuellos de botella en el gestor primario.

Servicios de tolerancia a fallas basados en replicación

Replicación activa

- En la replicación activa todos los gestores de réplicas tienen el mismo rol. Los frontales multidifunden las peticiones a todos los gestores y todos los frontales procesan la petición de manera idéntica pero independiente



Servicios de tolerancia a fallas basados en replicación

Replicación activa

- Al igual que la replicación pasiva, una replicación activa tiene las siguientes
  - Petición.
  - Coordinación.
  - Ejecución.
  - Acuerdo.
  - Respuesta.

Servicios de tolerancia a fallas basados en replicación

Replicación activa

- Durante la fase de petición, el frontal multidifunde la petición a los gestores usando multidifusión fiable y de ordenación total. No se envía otra petición hasta que se reciba la respuesta a la petición actual. El sistema de comunicación durante la fase de coordinación entrega la petición a todos los gestores según una ordenación total. Entonces, cada gestor realiza la fase para ejecutar la petición. La fase de acuerdo no es necesaria, debido a que se utiliza multidifusión. Finalmente, en la fase de respuesta, cada gestor manda su respuesta al frontal. El número de respuesta que recibe el frontal está en función de las asunciones de falla y del algoritmo de multidifusión.

Servicios de tolerancia a fallas basados en replicación

Replicación activa

- En la replicación activa, la tolerancia a fallas está soportada principalmente por la multidifusión fiable y totalmente ordenada. La multidifusión es equivalente a un algoritmo de consenso, por lo que la replicación activa permite resolver fallas bizantinas. Esto se debe a que el frontal recoge f+1 respuestas iguales antes de responder al cliente.

Consistencia

- La consistencia en los sistemas de cómputo es una necesidad vital, ya que sin ella el sistema simplemente no funciona. El problema de la consistencia está presente tanto en los sistemas de cómputo centralizados como en los sistemas distribuidos. El acceso a los datos en un sistema centralizado puede caer en estado inconsistente ante accesos concurrentes, por lo que se requiere prever mecanismos de exclusión que eviten estos escenarios. Sin embargo, en los sistemas distribuidos el problema de la inconsistencia es de mayor dimensión, tanto por su importancia como por la gran cantidad de situaciones en que puede producirse.

Consistencia

- Debido a que un sistema distribuido es un único sistema, entonces debe tener un único estado global compartido por todas las computadoras que la componen. Este estado global comprende las tablas de mantenimiento del sistema, la hora actual o los datos que están siendo compartidos por distintas computadoras.

Tipos de inconsistencias

En un sistema distribuido, los diferentes tipos de consistencias más frecuentemente estudiadas son:

- Consistencia de actualización.
- Consistencia de replicación.
- Consistencia de caché.
- Consistencia de reloj.

Consistencia de actualización

- La consistencia de actualización es importante observarla cuando varios procesos acceden concurrentemente a un dato para actualizarlo, ya que la actualización de todo el dato en su conjunto no se realiza como una única operación atómica y se puede caer en una inconsistencia de actualización. Este tipo de inconsistencia es un problema clásico en el acceso a bases de datos o datos compartidos. Sin embargo, en los sistemas distribuidos esto tiene una mayor relevancia debido a que estos sistemas generalmente tienen un gran número de usuarios. Este tipo de inconsistencia se evita usando *transacciones*.

Consistencia de actualización

- Las transacciones son las primitivas equivalentes a las entradas y salidas de una región crítica usada para proteger la consistencia de un dato compartido. Asimismo, una transacción asegura que las operaciones incluidas en esta se ejecuten todas o no se ejecuta una sola.

Consistencia de replicación

- Una situación de inconsistencia de replicación puede producirse cuando un conjunto de datos debe replicarse en diversas computadoras de la red, pudiendo ser modificado por cualquiera de estas. En este escenario, es muy probable que se puedan producir situaciones en que los datos no sean iguales en todas las computadoras al mismo tiempo. Los juegos interactivos multiusuarios pueden ser un ejemplo de inconsistencia de replicación, pues las acciones que introduzca un jugador deben de propagarse usando un protocolo de difusión de manera inmediata al resto de los jugadores en la red.

Consistencia de replicación

- En caso de que la red falle o tenga alta latencia, cada jugador tendrá una visión distinta del juego. La consistencia de replicación tiene gran importancia en los sistemas distribuidos, más aun si estos son en ambientes colaborativos e interactivos.

Consistencia de caché

- Cuando una aplicación cliente accede a archivo de datos, se pueden guardar estos datos en un espacio de la memoria local del lado del cliente para facilitar el acceso a este dato en futuras referencias. Esto se conoce como memoria caché y reduce la transferencia de datos por la red. La memoria caché tiene como objetivo mejorar los accesos locales mediante el modelo de jerarquías de memoria [Patterson & Hennessy, 1994]. Sin embargo, el problema de inconsistencia se puede presentar cuando el cliente también tiene que actualizar el mismo dato pero este reside en las memorias cachés de otros clientes.

Consistencia de caché

- En este caso, existe el riesgo de que las copias del dato en las otras memorias cachés queden desactualizadas. Para evitar este escenario, se deben considerar técnicas que aseguren la consistencia de los cachés por parte de los sistemas operativos distribuidos o por las arquitecturas de sistemas multiprocesadores.

Consistencia de reloj

- Diversas aplicaciones y programación en ambientes distribuidos basan su funcionalidad en algoritmos que muchas veces dependen de estampillas de tiempo. Estas estampillas de tiempo son usadas para indicar el momento en que un evento ha sucedido. Por ejemplo, algunos algoritmos de reemplazo de página en memoria virtual hacen uso de estampillas de tiempo. Una computadora en un sistema distribuido puede generar una estampilla de tiempo, la cual se puede enviar a cualquier otra computadora del sistema.

Consistencia de reloj

- Así, cada computadora puede comparar su estampilla de tiempo local con la estampilla de tiempo recibida. El problema de inconsistencia de reloj se presenta debido a que no es fácil mantener la misma hora física en todas las computadoras del sistema de manera simultánea.

Modelos de consistencia

- Los modelos de consistencia para los datos compartidos son a menudo difíciles de implementar de manera eficiente en los sistemas distribuidos a gran escala [Tanenbaum & Van Steen, 2008]. Además, en muchos casos se pueden utilizar modelos más simples, que también son a menudo más fáciles de implementar. Un modelo de consistencia es un contrato en los procesos y el almacenamiento de datos. Es decir, si los procesos acuerdan obedecer ciertas reglas, entonces el almacenamiento promete trabajar correctamente. Normalmente, una operación de lectura debe retornar la última actualización del dato. Los modelos de consistencia pueden ser:
  - Centrados en los datos.
  - Centrados en el cliente.

Modelo de consistencia centrado en los datos

- Este modelo asume que un almacén de datos (base de datos distribuida o un sistema de archivos) puede estar físicamente distribuido en varias máquinas. Todo proceso que pueda acceder a datos del almacén tiene una copia local disponible de todo el almacén y todas las operaciones de escritura se propagan hacia las otras copias. Ante la ausencia de un reloj global, es difícil sincronizar y determinar cuál es la última operación de escritura.

Modelo de consistencia centrado en los datos

Modelos de consistencia que no usan variables de sincronización:

- Estricta.
- Linealizada.
- Secuencial.
- Causal.
- FIFO.

Modelos de consistencia con operaciones de sincronización:

- Débil.
- Relajada.
- Entry.

Modelo de consistencia centrado en el cliente

- Este modelo de consistencia está orientado a un conjunto de datos que tienen un bajo número de actualizaciones simultáneas o, cuando estas ocurren, pueden resolverse fácilmente. Generalmente está orientado a operaciones de lectura de datos, por lo que se puede catalogar como un modelo de consistencia bastante débil [Tanenbaum & Van Steen, 2008]. Los modelos de consistencias basados en el cliente permiten ver que es posible ocultar muchas inconsistencias de un sistema distribuido de una manera relativamente fácil.

Modelo de consistencia centrado en el cliente

- Tanenbaum y Van Steen [2008] citan como ejemplos de modelos de consistencia centrados en el cliente los siguientes:
  - Consistencia momentánea.
  - Lecturas monotónicas.
  - Escrituras monotónicas.
  - Lea sus escrituras.
  - Las escrituras siguen a las lecturas.

Tolerancia a Fallas

- Los sistemas distribuidos se distinguen principalmente de los sistemas centralizados de una sola computadora en su capacidad de falla parcial. La tolerancia a fallas ha sido un tema de investigación por muchos años en las ciencias de la computación. Un sistema tolerante a fallas es un sistema que posee la capacidad interna para asegurar la ejecución correcta y continuada a pesar de las presencia de fallas en hardware o software. El objetivo de este tipo de sistemas es ser altamente fiable.

Origen de una falla

El origen de las fallas en un sistema puede ser clasificado como:

- Fallas en hardware:
  - Son generalmente fallas permanentes o transitorias en los componentes del hardware.
  - Fallas permanentes o transitorias en los subsistemas de comunicación.
- Fallas en software:
  - Se originan por especificaciones inadecuadas.
  - Fallas introducidas por errores en el diseño y programación de componentes de software.

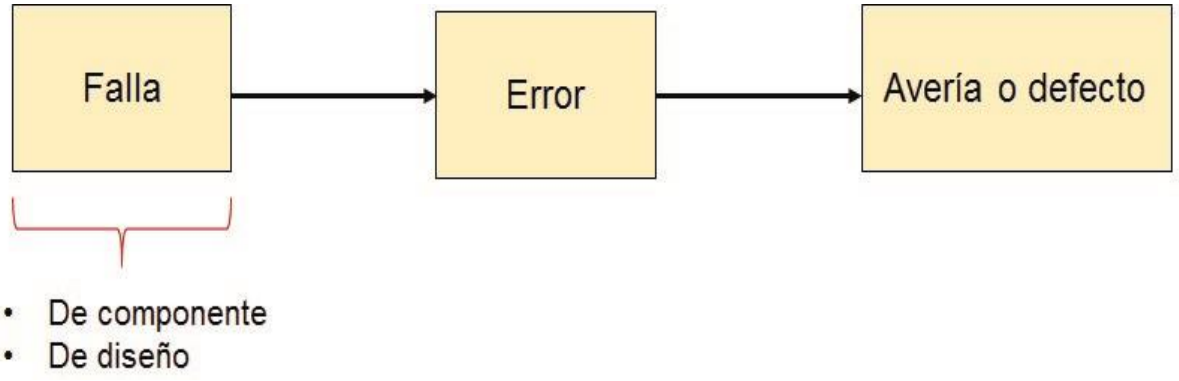
Origen de una falla

Se dice que un sistema distribuido es un sistema fiable cuando comprende varios requerimientos útiles, como los siguientes:

- Disponibilidad.
- Confiabilidad.
- Seguridad.
- Mantenimiento.

Origen de una falla

- La fiabilidad de un sistema es una medida de su conformidad con una especificación autorizada de su comportamiento [Tanenbaum & Van Steen, 2008]. En contraste, una avería o defecto es una desviación del comportamiento de un sistema respecto a esta especificación. Las averías se manifiestan en el comportamiento externo del sistema pero son el resultado de errores internos. Las fallas pueden ser consecuencia de averías en los componentes del sistema. Las fallas pueden ser pequeñas pero los defectos muy grandes.



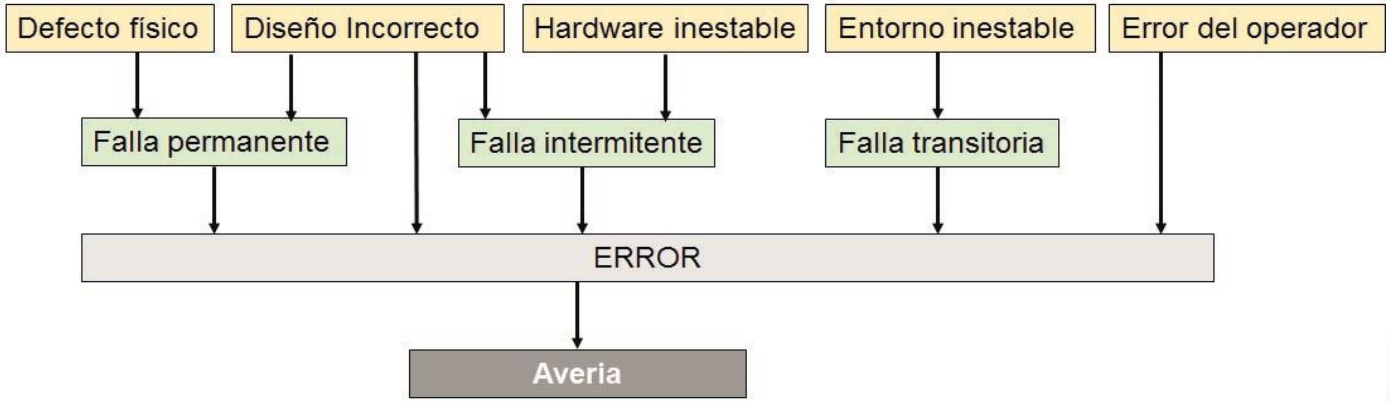
Clasificación de fallas

El objetivo de la tolerancia a fallas en un sistema distribuido es evitar que las fallas produzcan averías. Transitorias.

- Intermitentes.
- Permanentes.

Clasificación de fallas

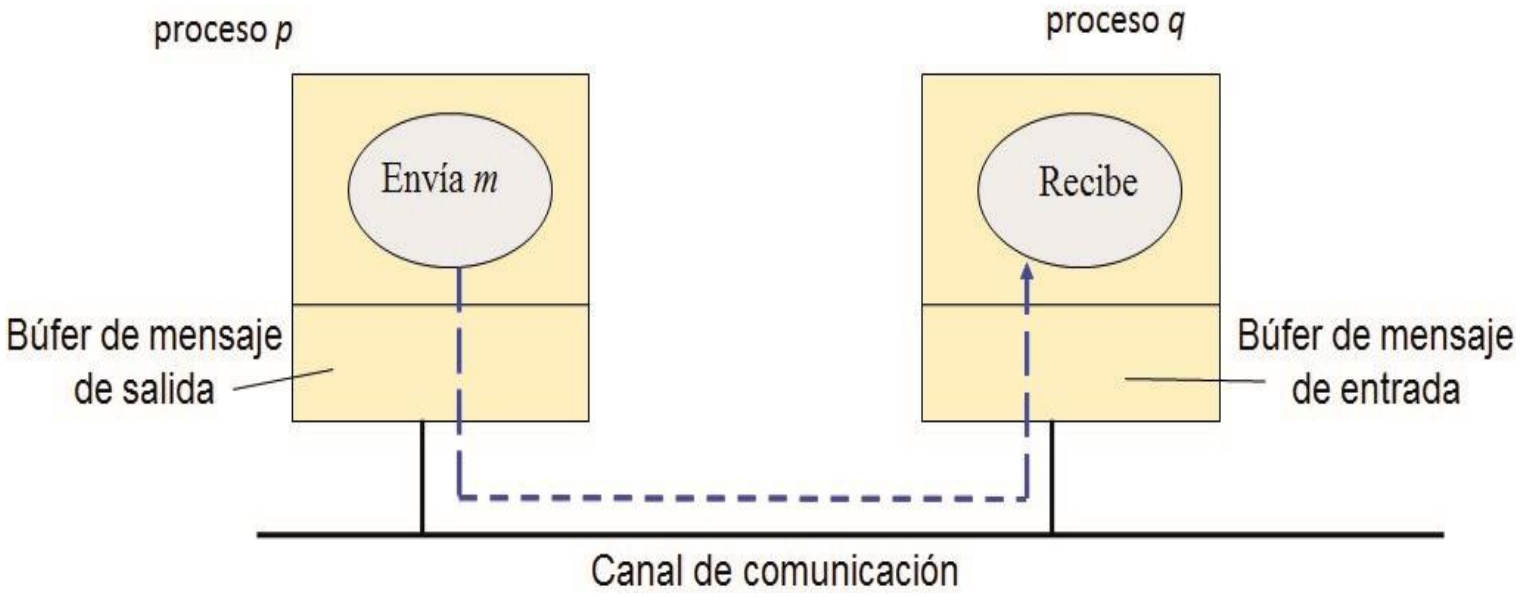
Es una falla transitoria, si esta desaparece sola al cabo de cierto tiempo. Ejemplos de este tipo de falla son las interferencias en las señales de comunicación o las fallas transitorias en los enlaces de comunicación. Por otro lado, las fallas permanentes son aquellas fallas que permanecen hasta que el componente se repare o sustituya, por ejemplo, los errores de software o la rotura de una tarjeta de video. Finalmente, las fallas intermitentes se refieren a aquellas fallas transitorias que ocurren de vez en cuando, por ejemplo, el calentamiento de algún componente de la computadora



Fallas en los procesos distribuidos

- En un sistema distribuido, tanto los procesos como los canales de comunicación pueden fallar, es decir que pueden apartarse de lo que se considera el comportamiento correcto o deseable. El modelo de fallas define las formas en las que puede ocurrir una falla, con el fin de proporcionar una comprensión de los efectos de los fallas. Bajo este enfoque, Hadzilacos y Toueg [1994] proporcionaron una taxonomía que distingue entre las fallas de los procesos y canales de comunicación y los presenta como fallas por omisión, fallas arbitrarias y fallas temporales

Fallas en los procesos distribuidos



Fallas en los procesos distribuidos

Tipo de falla	Efecto	Descripción
Falla de congelación	Proceso	El proceso para y permanece así. Otros procesos pueden detectar este estado
Crash (caída)	Proceso	El proceso para y permanece así. Otros procesos pueden no ser capaces de detectar este estado
Omisión	Canal	Un mensaje insertado en un buffer de mensajes de salida no llega al siguiente buffer de llegada de mensajes
Omisión de envío	Proceso	Un proceso completa un envío pero el mensaje no es puesto en su buffer de mensajes de salida
Omisión de recepción	Proceso	Un mensaje es puesto en el buffer de mensajes de llegada de un proceso pero este no lo recibe
Arbitraria (Bizantino)	Proceso o canal	El proceso/canal muestra un comportamiento arbitrario: podría enviar/transmitir arbitrariamente mensajes en tiempos arbitrarios, comete omisiones; un proceso puede detenerse o tomar un paso incorrecto

Fallas en los procesos distribuidos

- Las fallas más serias son las arbitrarias o bizantinas. Cuando estas fallas ocurren, los clientes deben de estar preparados para lo peor. Este problema suele conocerse en la literatura como el problema de los Generales Bizantinos, que fue planteado originalmente por Lamport et al. [1982].

Redundancia

Todas las técnicas de tolerancia a fallas se basan en el uso de la redundancia, de la cual pueden ser identificados los siguientes tipos:

- Redundancia estática: Los componentes redundantes se utilizan dentro del sistema para enmascarar los efectos de los componentes con defectos.
- Redundancia dinámica: La redundancia se utiliza solo para la detección de errores. La recuperación debe de realizarla otro componente.
- Redundancia en el diseño: Partes de un diseño pueden ser redundantes.
- Redundancia temporal: Mediante el uso repetido de un componente en presencia de fallas. Adecuado para fallas transitorias.

- Sistemas distribuidos / Francisco de Asís López Fuentes. -- México : UAM, Unidad Cuajimalpa, c2015