Análisis de la tolerancia de fallos en sistemas distribuidos

Jose Luis Martínez Ortiz

[1] Universidad de Granada
[2] Escuela Técnica Superior de Ingeniería Informtica y Telecomunicación

Abstract. Este trabajo consta del del análisis de los fallos y su tolerancia en los Sistemas distribudos, apoyandome en articulos donde destacan documentos como los de "*Gartner*" [2], aunque tenga ya unos aos sigue siendo muy ilustrativo y esclarecedor sobre el tema, definiendo y clasificando distintos tipos de fallos y como de graves pueden ser, que veremos más adelante. También hay otros trabajos ms actuales ofreciendo soluciones modernas a este problema clásico de los sistemas distribuidos, como diseos de Arquitecturas Software u otras técnicas.

1 Introduccin

Los sistemas distribuidos deben contar con una serie de requisitos para que se pueda considerar el sistema como un sistema distribuido, es decir, estos requisitos no están adheridos al sistema distribuido por si solo, si no que un sistema distribuido debe satisfacer unas caractersticas básicas. Para ello hay que disear el software y el hardware para que cumplan dichos requisitos, y para esto nos podemos ayudar de diversas técnicas para conseguir sistemas distribuidos. Las caractersticas u objetivos que debe conseguir un sistema distribuido son:

- Compartición de recursos.
- Sistemas abierto.
- Concurrencia.
- Escalabilidad.
- Tolerancia a fallos.
- Transparencia.

Este trabajo se centra sobre la ***Tolerancia a fallos****. El motivo de que la tolerancia de fallos sea una caracterstica básica de un sistema distribuido es que para este tipo de sistemas la fiabilidad y la "confianza" del sistema es primordial para el correcto funcionamiento. Si en un sistema distribuido falla alguna parte de él por pequea que sea puede causar el fallo completo del sistema, provocando severos daos. Por todo esto el análisis de los fallos de un Sistema Distribuido es tan importante. Aunque el proceso de disear un software a la parte de la tolerancia de fallos se suele dejar relegada a un segundo plano o no se le presta tanta atención como al mantenimiento, acoplamiento de los módulos, etc, debera ser una parte fundamental por las necesidades anteriormente explicadas.*

Según "Gartner" en su artículo que trata en concreto los sistemas asíncronos, ya que como comenta este tipo de sistemas son más débiles en la tolerancia de fallos con respecto a los sistemas síncronos , un detalle a tener en cuenta a la hora de elegir el modelo del sistema.

2 Definición

Lo primero es definir la nomenclatura que se utiliza normalmente con respecto a los fallos en un sistema, y se define mediante la estadística y de dos formas principalmente, ámbas no excluyentes entre si. La primera es una probabilidad de que ocurra un determinado tipo de fallo en el sistema, también conocido en ingles como "mean time to failure". La segunda y no menos importante la diferencia de tiempo entre un fallo que ocurre en el sistema y un fallo anterior del mismo tipo o componente.

La tolerancia de fallos en sistemas se componen de dos partes: diseño de los posibles fallos que pueden ocurrir en el sistema y la evaluación y comprobación de que el sistema tolera los fallos para los que se ha diseado. Felix "Gartner" se centra en el diseo de tolerancia de fallos, no en la fase de evaluacin de fallos, dado que en esta fase realizar los cambios es menos costoso y más rápido que en otras fases posteriores. El diseo de la tolerancia de fallos no es fácil ya que requiere de una gran experiencia en este ámbito y una gran visió de futuro. Algunos posibles fallos son previsibles, como el paso de datos entre mdulos o fallos de tipos de variables por ejemplo, pero no todos los fallos son tan fáciles de ver a priori. El desconocimiento o la falta de experiencia del diseador puede producir una falta de precisin en la definicin de posibles fallos del sistema y por consecuencia se produce una falta para disear una solucin a dichos fallos, provocando que la tolerancia a fallos del sistema sea incompleto o bastante deficiente generando de esta forma software de baja calidad.

Un tema importante dentro de la tolerancia a fallos es que estamos en un sistema distribuido y no debemos olvidarnos de que la comunición principalmente se realiza mediante mensajes entre los distintos ódulos del sistema, dichos módulos pueden estar alojados en distintas máquinas o no. La forma de comunicación se tiene que tener muy en cuenta para el diseo de la tolerancia de fallos. Las comunicaciones pueden ser tanto enviar como recibir eventos externos o internos se debe abstraer completamente de la red, ya que ser la red la encargada de controlar sus fallos, pero a nivel de aplicacin, del modelo TCP/IP, debe ser el propio sistema el encargado de dicho control.

2.1 Modelos de fallos

Un fallo puede terner tiene múltiples interpretaciones, como un defecto al más bajo nivel de abstracción, o consecuencias para el sistema que utiliza dicho software. Un fallo puede causar distintos tipos de error dependiendo del estado del sistema gracias a esto podemos recubrirlos con una capa de proteccin para que el sistema pueda tolerar dichos fallos. Como "Gartner" explica en su artículo, define principalmente tres tipos de fallos:

1. ***crash**: Este tipo de fallo provoca que la unidad de computo deje de funcionar, sin un motivo aparente.*

2. ***fail-stop****: Cuando el software local deja de funcionar. Entendemos por software local los módulo acoplados y aquellos de los que depende el módulo afectado.*
3. ***fallo-variable****: Se produce un cambio no contemplado en un estado del sistema, que deriva en una situacin no deseada.*

Es importante esta clasificacin, ya que nos permite identificar mejor la búsqueda de posibles fallos durante la fase de diseo, que es la más costosa. Aun así el fallo de tipo "crash" son muy difíciles de detectar, por que es un fallo que no siempre ocurre, si no bajo unas circunstancias muy concretas y es complicado de replicar. Este fallo suele costar mucho tiempo en detectarse cual es su causa produciendo retrasos en el sistema y aumentando los costes del mismo.

2.2 Tolerancia a fallos

Una vez que hemos definido que són los fallos y hemos comentado en varias ocasiones la "tolerancia a los fallos", pero todavía no se había definido.

La ***Tolerancia a fallos*** *o fault tolerance es la propiedad o habilidad de un sistema software de comportarse de una manera pre-establecida cuando ocurre un fallo.*

Una vez definido que es la tolerancia a fallos, explicaremos en que consiste. Lo primero durante la fase de diseo del sistema es encontrar y definir los fallos en sus distintas fases, como se explicó anteriormente. Por último Una vez recopilados todos los posibles fallos y localizados hay que listar las posibles soluciones a cada uno de ellos y evaluar si con dicha solucin se solventa el fallo o por lo menos se contiene para que no produzca un fallo más grave y problemático.

En los sistemas de software tenemos dos propiedades básicas, y dependiendo del tipo concreto de sistema una será más importante que la otra, hablamos de la propiedad de seguridad y disponibilidad. Si una aplicacin no esta nunca disponible nadie puede utilizarla pero también dejarán de utilizar el sistema si esta disponible de forma intermitente impidiendo su correcta utilización. Estas propiedades nos pueden ayudar a la hora de clasificar los tipos de tolerancia de fallos que vamos a implementar en el sistema y tener otra forma de evaluar la prioridad a la hora de disear mecanismos de tolerancia para un determinado fallo u otro. Para ello clasificamos las tolerancias a los fallos que hemos detectado. Y suponiendo las propiedades que hemos citado en este párrafo obtenemos cuatro combinaciones o niveles de tolerancia de fallos posibles cuando ocurre un fallo.

	Disponible	No disponible
Seguro	enmascarado	fallo seguro
No seguro	brecha	nada

El tipo ***enmascarado*** *ocurre cuando un fallo se controla y no produce una brecha en la seguridad del software ni pone en peligro su disponibilidad. Es el mejor tipo posible pero es la solución que requiere más tiempo y dinero.*

*El tipo **fallo seguro** protege la seguridad del sistema a costa de la disponibilidad, la cual no se garantiza. Pero evitamos que se puedan producir fallos de seguridad que sean irreparables.*

*El tipo **brecha** es el peor recubrimiento a un fallo posible, ya que permitimos que el fallo que produzca una brecha en la seguridad del sistema siga activo puesto que se garantiza la disponibilidad del sistema, pudiendo dar lugar a estados indeseables irreversibles y muy peligrosos para algunos sistemas. En alguna situaciones es preferible no recubrir o tolerar este fallo y dejar que el sistema caiga, bloqueando así dicha brecha en la seguridad.*

El ultimo tipo es el "nada" y como su propio nombre indica no hay ninguna tolerancia a fallos.

2.3 Soluciones generales

Las formas de tolerar un fallo son dos, la duplicidad de los datos o redundancia y las soluciones software.

Una solucin es la redundancia y que es la ms usada, tanto para los recursos hardware como los recursos software, básicamente consiste en tener un sistema distribuido redundante. Pero qué es la Sistema distribuido redundante? Si tomamos la definición de [2] podemos definirlo como: "Un programa distribuido A se dice que es redundante en el espacio si para todas las ejecuciones e de A en las que no hay faltas y el conjunto de todas las configuraciones de A contiene configuraciones que no se alcanzan en e." Otra forma de interpretar esta definición es: se dice que A sea redundante en el tiempo si para todas e ejecuciones de A contiene acciones que nunca se ejecutan en e.

La otra solució es la recuperación software, que permite recuperar la falta de información o tolerar un fallo mediante la utilización de mecanismos softwares. Algunos ejemplos de estos mecanismos son, utilizar los invariantes de representación y poner medidas que obliguen al sistema a no salirse de los invariantes, capturar los errores de forma manual y tratarlo de forma segura y controlada. También es cierto que este tipo de solución es la más compleja ya que requieren de otros mecanismos que revisen la información y sepan cómo pueden reparar el fallo. En algunos casos esta tarea es imposible puesto que la informacin para poder ser recuperada mediante software es necesario que incorpore informacin extra o datos de seguridad para indicar cómo recuperar la informacin. Esta forma tiene las grandes desventajas, como que aumenta la computació en todo el sistema ya que requiere que se análice siempre toda la informacón extra para resolver el fallo y ademá solo funciona con los recursos software, si ocurre un fallo en una unidad de almacenamiento no es posible recuperar el contenido de la unidad.

3 Arquitecturas Software

Tradicionalmente se seguían dos corrientes para la tolerancia a los fallos, el soporte global al sistema para resolver la tolerancia o un soporte individual a los módulos. Esta tendencia ha ido cambiando con forme se mejoraba la fase de diseo de los sistemas software y aparecían nuevas formas de disear sofware más dinámicas y rápidas que las clásicas. De esta forma en la tolerancia a fallos apareció el termino "smart redundancy" que busca la redundancia de los datos pero de una forma inteligente para que la duplicidad de los datos sea la mínima posible y así aprovechar mucho mejor los recursos del sistema. Una forma de diseo que ha recogido esta idea y muchas otras es la Arquitectura de Software que ha sido desarrollada hace relativamente muy poco y que está teniendo una gran acogida por la comunidad de la Ingeniería Software. Hoy en día es la base para casi todo el software, incluido el distribuido y esto es debido a las grandes ventajas que nos ofrece.

Como se comentó en la sección anterior las soluciones software son complejas de aplicar y de disear pero nos podemos valer de esta herramienta que es muy potente. Tal y como comenta Brun [1] en su artículo las Arquitecturas Software ya han solucionado algunos problemas de diseo de los sistemas software y de una forma eficiente y consensuada. Es por esto que utilizar arquitecturas que favorezcan, fomenten o premien la tolerancia a los fallos provoca que sea más fácil la resolución de medidas software que hasta hace poco erán aunténticos rompecabezas y complejas sus soluciones.

4 Análisis

En general la tolerancia a los fallos se tiene muy en cuenta en el desarrollo del software en general, pero bien es cierto que en sistemas distribuidos donde un sistema software está repartido por diferentes zonas espaciales la comunicación entre sus componentes es esencial y la forma de garantizar las comunicaciones y el manejo de la información debería de ser de las primeras a tener en cuenta.

Es curioso como a finales del siglo pasado ya había preocupación por los fallos que producían los sistemas y hay una gran cantidad de información en esa época. Muchos de esas ideas son la base para los algoritmos actuales o incluso son los mismos mecanismos los que se utilizan para el control de los fallos. En el caso de Felix Gartner en su articulo, en el cual me he basado principalmente, hace una clara y magnifica recopilación de los mejores artículos de la fecha y los muestra junto a su experiencia. La forma de analizar y clasificar los tipos de fallos son fundamentales para poder tratarlos de forma específica y consiguiendo unas medidas de control muy particulares de otros que requieren otro tipo de medidas obteniendo un resultado más satisfactorio y rápido de logra. De esta forma agrupamos los fallos que requieren las mimas medidas y las aplicamos de una sola vez.

Con respecto a las soluciones existe un gran debate entre tener los recursos redundantes o buscar las soluciones mediante software más óptimas. Si se elige la redundancia nos aseguramos cierta tolerancia a los fallos, pero quiere un gran desembolso económico, espacial y personal en comprar nuevos equipos, prepararlo y manterlos funcionando y listo por si ocurre un fallo. O Invertir personal y dinero en la búsqueda de soluciones software que no garantiza que se encuentren y de ser encontradas, hay que ver su viabilidad en el sistema y funcionamiento. Por ello es un eterno debate que parece que se está solucionando de forma híbrida gracias a los nuevos diseos. La Arquitectura de software está reduciendo o calmando las principales desventajas de las soluciones no redundantes pero siguen sin ser efectivas al 100% y con la computación distribuida y aumento de componentes gracias a la industria electrónica se está minimizando el coste de duplicar los recursos hardware.

Como conclusión final considero que es una linea de investigación donde apenas se ha avanzado mucho y que tiene un largo recorrido por delante. Considero que seguir buscado nuevas formas de tolerancia de fallos software y que compense la falta de un dispositivo hardware sería un gran avance en este sentido y permitirá avanzar en gran medida a los sistemas softwares más complejos.

5 Referencias

References

1. Yuriy Brun and Nenad Medvidovic. Fault and adversary tolerance as an emergent property of distributed systems' software architectures. In *Proceedings of the 2007 Workshop on Engineering Fault Tolerant Systems*, EFTS '07, New York, NY, USA, 2007. ACM.
2. Felix C. Gärtner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Comput. Surv.*, 31(1):1–26, March 1999.