

**Tecnológico Nacional de México**

**Instituto Tecnológico de Reynosa**

Materia: Sistemas Operativos 2

ACTIVIDAD09: Investigación exhaustiva del Tema04: Tolerancia a fallas con sus 4 subtemas.

Tema 4: Tolerancia a fallas

Alumno: Castillo Jr. Gregorio

Numero de control: 19580589

Correo electrónico: L19580589@reynosa.tecnm.mx

7mo Semestre Matutino Salón 7

Docente: Mario José Santiago Sánchez

Fecha de entrega: 22/11/2022



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Consecutivo** | **Numero de**  **control** | **Apellido Paterno** | **Apellido Materno** | **Nombres(s)** | **Correo electrónico**  **Institucional** | **Firma de que está**  **autorizando que se**  **entregue esta Publicación** | **Fotografía del rostro de cada**  **Integrante del equipo (selfie)** |
| **1** | **19580589** | **Castillo Jr** |  | **Gregorio** | **L@19580589@reynosa.tecnm.mx** |  |  |
| **2** | **19580595** | **Flores** | **Acosta** | **Sheila Lizeth** | **L@19580595@reynosa.tecnm.mx** |  |  |
| **3** | **19580867** | **Morales** | **Calixto** | **Daniel Alexander** | **L@19580867@reynosa.tecnm.mx** | No hay descripción disponible. |  |

Equipo #12

# Tabla de contenidos

[Tabla de contenidos 3](#_Toc120037254)

[Contenido 4](#_Toc120037255)

[**3.1 Tolerancia a fallas** 4](#_Toc120037256)

[**3.2 Introducción atenuación de un proceso** 6](#_Toc120037257)

[**3.3 Comunicación confiable cliente-servidor** 14](#_Toc120037258)

[**3.4 Comunicación confiable en grupo** 19](#_Toc120037259)

[**3.5 Recuperación** 26](#_Toc120037260)

[**Conclusiones** 34](#_Toc120037261)

[**Herramientas y recursos** 35](#_Toc120037262)

[**Bibliografía** 36](#_Toc120037263)

# Contenido

## **3.1 Tolerancia a fallas**

La capacidad de un sistema para funcionar normalmente a pesar de interrupciones repentinas se denomina tolerancia a fallos. En las últimas décadas, la ciencia de la ingeniería de hardware y software ha logrado enormes avances que han afectado a toda la comunidad de ingenieros. En particular, innovaciones como computadoras con múltiples núcleos, unidades flash con mayor capacidad de almacenamiento y redes inalámbricas con mayor velocidad han cambiado nuestra vida cotidiana. Gracias a estas mejoras en la tecnología, también hemos creado mejores sistemas para ayudarnos a realizar tareas complejas. En esa medida, diseñamos nuestros sistemas para que sean tan tolerantes a fallas como sea posible para que duren mucho tiempo sin fallar.

La tolerancia a fallas es una medida de la capacidad de un sistema para llevar a cabo sus funciones normales en condiciones adversas. Por ejemplo, la tolerancia a fallas de un sistema informático limita el tiempo que puede funcionar normalmente después de experimentar una falla. Algunas formas comunes de aumentar la tolerancia a fallas de un sistema son las funciones redundantes, las copias de seguridad y los planes de conmutación por error. Los sistemas redundantes proporcionan una ruta alternativa para completar tareas cuando falla el sistema principal, sin comprometer las funciones críticas de la tarea. Además, se deben implementar planes de respaldo para que el sistema continúe funcionando incluso después de que falle el sistema principal.

A menudo falta tolerancia a fallas en los nuevos diseños o cuando se toman enfoques innovadores para aumentar el rendimiento. Esto se debe a que la innovación es un proceso rápido que conduce naturalmente a diseños actualizados con mayores capacidades de tolerancia a fallas. Por lo tanto, es mejor aumentar la tolerancia a fallas de un sistema aumentando los diseños y sistemas existentes en lugar de mediante la innovación. Esto se debe a que la innovación es un proceso dinámico y es posible que los nuevos diseños no estén disponibles cuando se necesiten, lo que aumenta aún más el riesgo de fallas en el sistema.

Al diseñar un plan de respuesta, los planificadores deben considerar las causas potenciales de las fallas y cómo manejarlas de manera efectiva. Por ejemplo, si el sistema principal falla debido a un corte de energía, el personal debe tener suficiente equipo de seguridad para funcionar durante 24 horas sin energía. Un buen enfoque de planificación garantiza que se hayan considerado todas las respuestas posibles antes de implementar cualquier plan de respuesta. De lo contrario, podría resultar en un plan ineficaz o completamente inútil que desperdiciará tiempo y recursos preciosos y pondrá en peligro tanto al personal como al equipo.

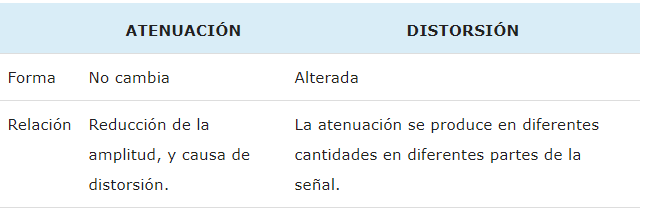
Un nivel robusto de tolerancia a fallas aumenta la capacidad de un sistema para funcionar normalmente a pesar de las interrupciones repentinas, lo que los hace ideales para llevar a cabo tareas complejas de manera eficiente. Esto incluye la planificación de respuestas y la preparación de sistemas redundantes para su uso cuando sea necesario, además de implementar soluciones innovadoras cuando estén disponibles. En última instancia, la mayor atención a las normas de seguridad en el mundo actual ha hecho posible que disfrutemos de posibilidades prácticamente ilimitadas en todos los campos de la actividad humana.

## **3.2 Introducción atenuación de un proceso**

Este tema nos demostrara el cómo se logra la tolerancia a fallas en sistemas distribuidos, tratando de tocar temas de cómo protegernos contra las fallas de proceso y una manera es replicando los procesos en grupos. Considerando los temas de diseño generales de grupos de procesos y analizamos el significado de un grupo tolerante a fallas.

Dentro de atenuación de procesos podemos encontrar puntos como:

* Forma clave de afrontar la tolerancia a un proceso defectuoso: organizar varios procesos idénticos en un grupo.
* La propiedad fundamental que tienen todos los grupos es que cuando un mensaje es enviado al grupo, todos los miembros de éste lo reciben.
* De esta manera, si en un grupo falla un proceso, afortunadamente alguno de los demás procesos puede hacerse cargo de él (Guerraoui y Schiper, 1997).



Sus características:

* Los grupos de procesos pueden ser dinámicos
* Se pueden crear grupos nuevos y destruir los viejos
* Un proceso puede unirse a un grupo o abandonarlo durante la operación del sistema.
* Un proceso puede ser miembro de varios grupos a la vez.
* En consecuencia, se requieren mecanismos adecuados para gestionar los grupos y la membresía a un grupo.

**TEMAS DE DISEÑO**

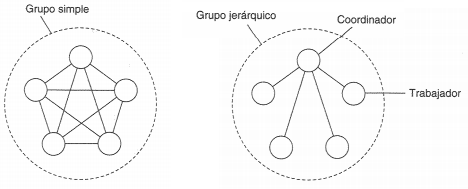
Una de las claves de afrontar la tolerancia a un proceso defectuoso es organizar varios procesos idénticos en un grupo ya que la propiedad fundamental que tiene todos los grupos es que cuando un mensaje se envía al grupo, todos los del grupo perteneciente reciben el mensaje también. De esta forma, si es que llega a fallar un proceso, afortunadamente alguno de los demás procesos se pueden hacer cargo de este.

Estos grupos pueden ser dinámicos, se pueden crear grupos nuevos y que los viejos queden en el olvido o destruirlos en palabras más claras. El proceso se puede unir a un grupo o abandonarlo durante la operación del sistema, pero necesitan mecanismos adecuados para gestionar los grupos y membresía a un grupo.

Para la explicación de los grupos, digamos que son más o menos como las organizaciones sociales. El propósito de esto es permitir que los procesos se ocupen de los conjuntos como una sola abstracción, por lo tanto, un proceso puede enviar un mensaje a un grupo determinado de servidores sin que deba saber cuáles son o cuantos son o donde están, lo cual pueden cambiar de una invocación a la siguiente.

Por eso, repitiendo de nuevo la mejor forma de enfrenar la tolerancia de un proceso es con la replicación de procesos en grupos.

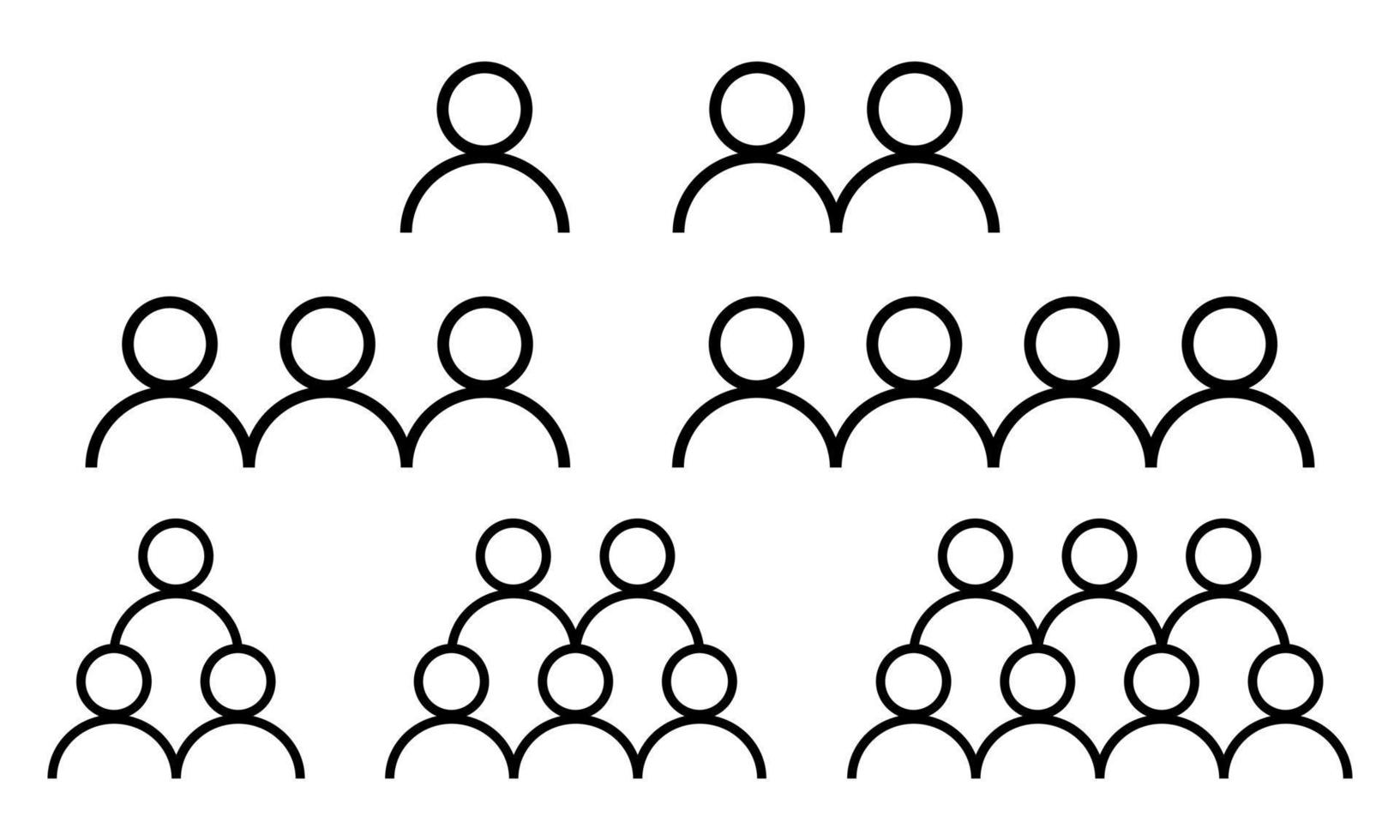
* Grupos dinámicos en su gestión.
* Abrir grupos nuevos y eliminar los viejos.



Existen dos tipos de grupos:

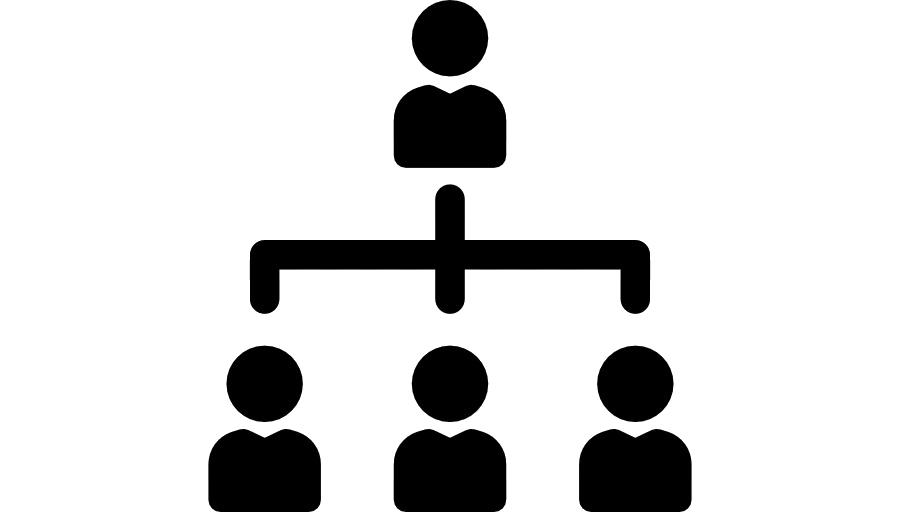
* Grupo simple

Si uno de los nodos se llega a congelar, el grupo se convierte más pequeño pero continua pero la desventaja es que la toma de decisiones es difícil. Por ejemplo, para poder decidir alguna cosa a menudo hay que someterlo a votación, eso implica retrasos y gastos indirectos.



* Grupo jerárquico

Aquí la perdida es del coordinador que hace que todo el grupo se detenga de manera brusca, pero muestra este funciona, puede tomar las decisiones sin molestar a los demás.



Una importante decisión entre los distintos grupos tiene que ver con la estructura interna. En algunos, todos los procesos son los mismos; ninguno es el que manda y todas las decisiones que se llegan a tomar de manera colectiva. En otros grupos, existe la jerarquía, por ejemplo; un proceso es el coordinador y los demás son los trabajadores. Este modelo es cuando un cliente externo o uno de los trabajadores genera una solicitud de trabajo y esta se envía al coordinador. El coordinador decide entonces cual de los trabajadores es el más apto para poder realizar el trabajo y lo remite allí, también son posibles jerarquías más complejas.

**Membresía a un grupo**

Cuando en algún grupo existe comunicación, se requiere de algún método para poder crear y eliminar los grupos, así como poder permitir que los procesos se unan a los grupos o los abandonen.

La comunicación en el grupo requiere un método para:

* Crear y eliminar grupos.
* Unión y separación de procesos a grupos.

Aquí existen dos métodos:

* Tener un servidor de grupos al cual estar enviando todas las solicitudes.

El servidor nos puede ayudar ya que puede mantener una base de datos completa de todos los grupos y la membresía exacta, suele ser un método eficiente y fácil de implementar, pero la desventaja es que el punto de fallo que representa la administración centralizada de los grupos.

* La administración distribuida de las membresías a grupos.

Este es un grupo abierto, donde un proceso extraño puede enviar un mensaje a los integrantes del grupo para poder anunciar su presencia. En un grupo cerrado se hace algo similar, ya que se debe contemplar la admisión de nuevos miembros al grupo cerrado.

Si se salen del grupo, el proceso se le debe de comunicar a los demás del grupo de que se saldrá.

Un aspecto problemático que suele presentarse cuando un miembro suele fallar, al salir, por lo tanto:

* No hay un anuncio apropiado de este hecho.
* Los demás miembros del grupo lo deben descubrir de forma experimental y luego se lo puede eliminar del grupo.

Otro aspecto de suma importancia es que la entrada y la salida al grupo se debe de sincronizar con el envió del mensaje:

* Un proceso que se unió a un grupo debe recibir todos los mensajes que se envíen al mismo.
* Un proceso que ha salido de un grupo: no debe recibir más mensajes del grupo, el grupo no debe recibir más mensajes del proceso y los otros miembros no deben recibir más mensajes del proceso saliente.

**ENMASCARAMIENTO DE FALLAS Y REPLICACIÓN**

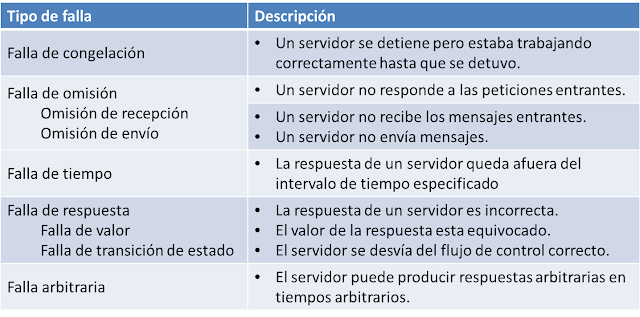
Los grupos de procesos son parte de la solución implementada que construye sistemas tolerantes a las fallas. Al tener un grupo de procesos idénticos permite disfrazar uno o más procesos defectuosos presentes a dichos grupos. En otras palabras, nosotros podemos replicar procesos y organizarlos en un grupo para reemplazar un solo proceso vulnerable con un grupo de procesos que sea tolerante a fallas.

**Replicación casada en un protocolo primario**

En el caso de fallas, generalmente aparece en forma de un protocolo de respaldo primario, así un grupo de procesos se puede organizar de forma jerárquica en la cual un protocolo primario coordina todas las operaciones de escritura.

**Protocolos de escrituras replicados**

Se utiliza en la forma de replicación activa, así como también por medio de protocolos. Estas soluciones corresponden a organizar un conjunto de los procesos idénticos en un grupo simple, una de las ventajas principales es que los grupos no tienen un solo punto de falla a causa de la coordinación distribuida.



**ACUERDO EN SISTEMAS DEFECTUOSOS**

La organización de los procesos replicados en un grupo incrementa la tolerancia a las fallas, es decir que, si un cliente puede basar las decisiones en un mecanismo de votación, incluso se puede tolerar que k de 2k + 1 procesos mienten sobre sus resultados. Es una suposición que se hace, sin embargo, es que los procesos no hacen equipo para producir un resultado equivocado.

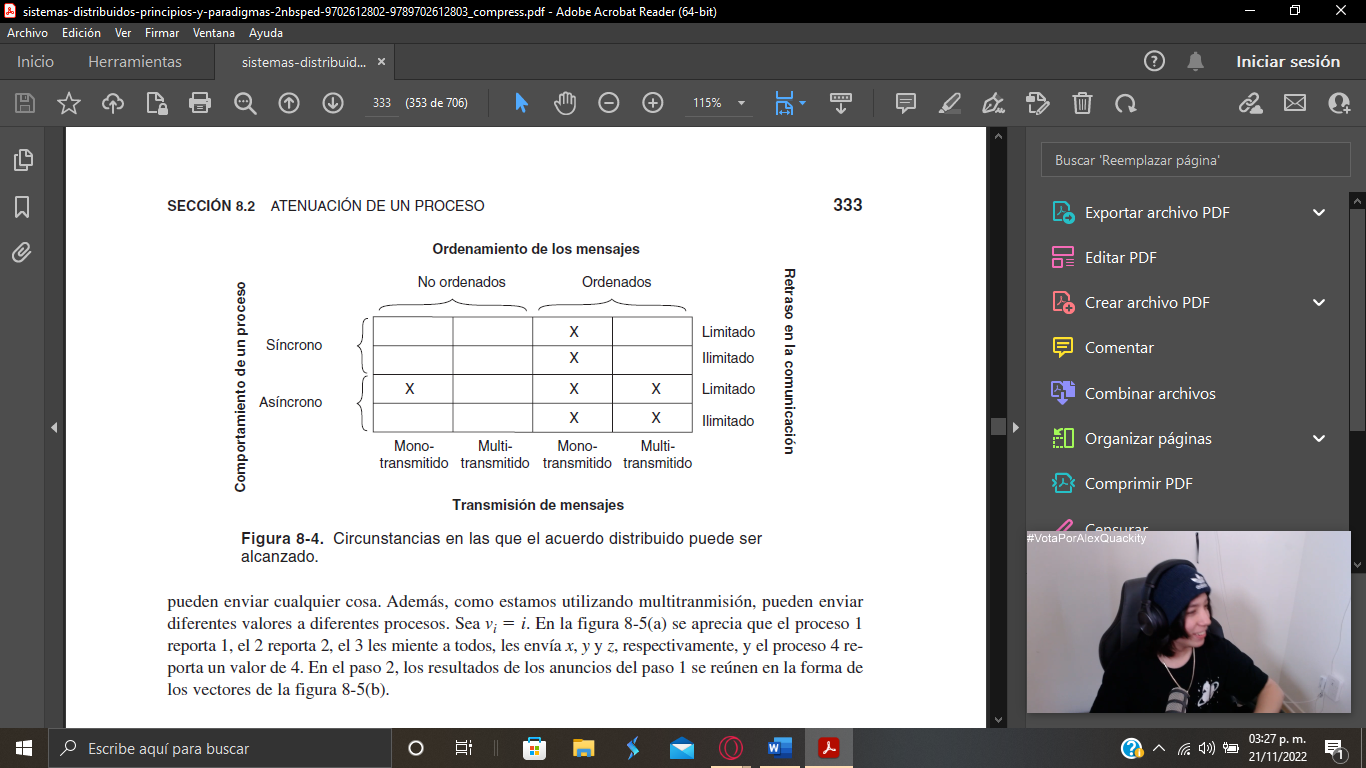
Las cosas se complican cuando se demanda que un grupo de procesos llegue a un acuerdo, lo cual es requerido en muchos casos.

El objetivo general de los algoritmos de acuerdo distribuidos es hacer que todos los procesos no defectuosos alcancen un consenso en un tema, y poder establecer dicho consenso dentro en algún numero finito de pasos. Pero se complica cuando las diferentes suposiciones sobre el sistema subyacente requieren diferentes soluciones.

Sistemas síncronos contra sistemas asíncronos:

1. El retraso de la comunicación está o no limitado
2. La entrega del mensaje se hace o no en orden
3. Los mensajes se transmiten unitransmisión o multitransmisión

Y solo es posible llegar a un acuerdo en la siguiente situación:



**DETECCIÓN DE FALLAS**

Para enmascarar las fallas es necesario detectarlas antes. La detección de fallas es una de las piedras angulares de la tolerancia a fallas en los sistemas distribuidos, a lo que esto se reduce para que en un grupo de procesos los miembros no defectuosos deberán de ser capaces de decidir quien sigue siendo miembro y quien no. De otra forma, debemos ser capaces de detectar cuando un miembro falla.

Cuando se detectan fallas de procesos, en esencia solo existen solo dos mecanismos o los procesos son enviados mensajes de manera activa del tipo ‘¿estas vivo?’ entre sí o esperan pasivamente hasta que lleguen mensajes de los diferentes procesos. Lo segundo tiene sentido solo cuando se garantiza que existe suficiente comunicación entre los procesos. Dentro de la practica generalmente se envía activamente un ping a los procesos.

La detección de fallas también puede ocurrir como un efecto colateral del intercambio regular de la información con los vecinos. Este enfoque también se adopta esencialmente en Obduro: los procesos son informados periódicamente acerca de su disponibilidad de servicio. Con esta información se disemina gradualmente por la red a través de charlas. Durante un tiempo cada proceso se enterará de la existencia de todos los demás procesos, pero lo más importante es que tendrá suficiente información localmente disponible para decidir si un proceso ha fallado o no. Un miembro para el cual la información sobre disponibilidad es obsoleta presumiblemente ha fallado.

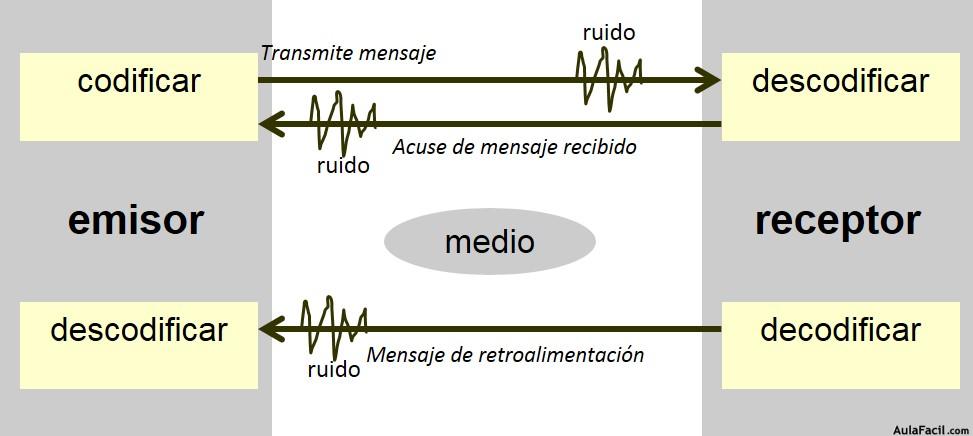
Esto conduce a otro tema: cuando se detecta la falla de un miembro ¿Cómo se debe infórmalo a otros procesos que no son defectuosos? Un método simple es que se sigue FUSE ya que todos los procesos pueden unirse en un grupo que corresponda a una amplia red. Los miembros crean un árbol expansivo para poder monitorear los miembros que fallan. Cuando un vecino no corresponde, el nodo que envió el ping cambia de forma rápida a un estado en el cual tampoco responderá a los pings enviados por otros nodos.

## **3.3 Comunicación confiable cliente-servidor**

Cuando hablamos de la comunicación cliente servidor esta no tiene más que en cómo es que conectamos un usuario con otro, es decir, todas las herramientas y procesos que utilizamos para poder trabajar lo que a una conexión remota se refiere, desde lo más simple como lo serían los cables para realizar las conexiones, hasta lo más complejo como los protocolos de comunicación previamente establecidos, es con ello que a lo largo de los años se han ido desarrollando nuevas tecnologías que nos permiten elaborar conexiones mucho más fiables y sobre todo más seguras, como sabemos, quienes se conectan a este tipo de software, pueden tener acceso no solo a lo que ellos necesitan si no a la información de otras entidades, personas o empresas, generando asi una brecha de seguridad que dependiendo la situación del afectado o la información recabada, puede ser mayor o menos el problema, de tal forma es que se han creado los modelos de comunicación donde encontramos distintas formas de mover, almacenar y utilizar la información

Modelos de comunicación

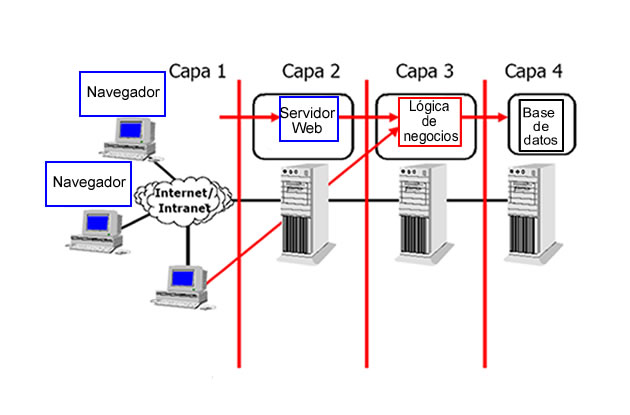
Existen muchos y muy diversos modelos de comunicación, a lo largo del tiempo estos se han ido modificando de acuerdo a la necesidad del usuario y del sistema en el que este se encuentra, dicho esto, también existen modelos que se han quedado obsoletos y hoy en día ya no se utilizan, esto porque han sido modificados, o se han creado versiones mejoradas de este.



El proposito de estos mensajes, como ya se sabe es, permitir la comunicación de información a lo largo de una vasta extensión territorial, y generando mensajes prácticamente instantáneos, del mismo modo para preservar la confidencialidad de estos mensajes se hacen uso de diversos protocolos como lo son el TCP/IP

Comunicación cliente-servidor

Esta comunicación es llevada a cabo por los modelos ya anteriormente mencionados, sin embargo cuando hacemos uso de estos modelos no basta con solo tenerlos, puesto que estos solo moderan el como es que se mandara la información que deseamos llegue al destinatario, sin embargo no contamos con las “conexión o parámetros” necesarios para hacer uso de estos modelos, es ahí donde entra en juego lo que se le conoce como codificación backend o dicho en otras palabras, la programación que hay detrás de la conexiones y nos permite enviar y recibir información, para ello hacemos uso de tecnologías como lo son las bases de datos, ayudándonos con SQL, realizar las conexiones con PHP, modelar interfaces como lo seria con HTML y generar utilidad con Java, esto radica en que no solo una tecnología sea involucrada a la hora de realizar una tarea tan compleja como lo es mandar un mensaje, también para ello es que tenemos que tener en cuenta a lo que el encadenamiento se refiere, que no es nada mas que el modelo de N capas, las cuales nos dicen en número de saltos de un servidor a otro para que se pueda realizar el envió de un mensaje, si bien existe la posibilidad de que este numero de capas aumente, siempre se buscara hacer la conexión por el numero de capas menor, esto puesto que supone una carga menos excesiva de recursos así como de tiempo, ya que de mandarse el mensaje por la capa más grande este puede perderse o llegar corrompido, debido a los múltiples saltos que dio el mensaje, perdiendo pequeñas fracciones de información en el camino



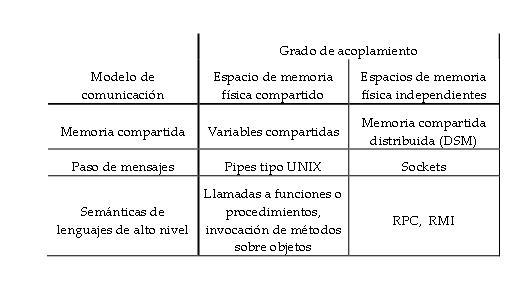
De este modo es que podemos ver como es que funcionan las capas, sirven para poder mandar el mensaje a travez de servidores, es ahi donde se hace uso de de los diferentes protocolos, de tal modo que nuestro sistema ya tiene la utilizacion de software por parte del cliente refiriendonos a php, html, etc. y tambien el uso del lado del servidor como lo son los protocolos TCP/IP



Es en esta tabla donde podemos ver de forma mas clara a lo que el utilizamiento de metodos hablamos, en esta vemos como actividades como el cloud computing que son ejecutadas por parte del servidor y del mismo modo son almacenados en esta, aquí podemos entender como es que funcionan las diversas tecnologias actuales y como de importantes son, debido a lo que esta actividad significa

Comunicacion por RPC (Remote call procedure)

claramente si hablamos de protocolos, uno de los mas importantes es el RPC o llamada remota, la cual nos permite tener acceso a un servidor, computadora o cualquier entidad de carácter informatica, permitiendonos interactuar con dicho dispositivo ya sea leyendo, modificando o agregando datos, dependiendo de los permisos con los que contemos, el proposito escencial de esta tecnologia no es mas alla de simplificar el metodo de administracion de datos en tecnologias remotas, evitando asi redireccionamientos innecesarios por parte del servidor local, y es aquí donde entra en juego lo que conocemos como memoraria compartida, la cual nos permite hacer modificaciones de carácter remoto en otra entidad

Tolerancia a fallas

Es el objetivo principal de un sistema distribuido y la razon por la cual fueron creados, cuando trabajamos con sistema unicos podemos ver reflejados multiples errores a la hora de trabajar con ello, esto debido a que los sistemas o funcionan o no lo hacen pues bien, los sistemas distribuidos permiten ejercer un llamado al rescate ya que al trabajar con nodos, existen muchas formas de llegar a un mismo resultado, ya que de encontrar fallas en una ruta, exiten n mas que nos permiten usarlar para resolver nuestro problema, sin embargo esto no quita que podamos tener fallos a la hora de trabajar con ellas, entre estos problemas encontramos 2 escenciales:

Asincrono: no hay cordinacion en el tiempo.

Sincrono: Se suponen limites maximos para el retraso del mensaje

para empezar con el Asincrono, este se refiere cuando existe un desface entre servidor A y servidor B, esto puede ser dado por mucahs situaciones, ingreso de actualizaciones, zona horaria, cambio de reloj, etc. esto traeria consigo el que los datos que ingresemos desde A hacia B no se vean reflejados en tiempo y forma ya que no se encuentra concordancia en las actualizaciones, y por cuestiones de seguridad el mensaje se pierda

El otro problema que se puede presentar es aque donde nos vemos limitados por l tiempo de conexión, de forma que si nos pasamos de este umbral, no nos sea permitido ni el acceso, ni la modificacion de ningun dato, ya que da a entender que quien esta ingresando puede ser alguien muy remoto y podria comprometer los datos que en dicho servidor se encuentra

Cabe recalcar en que estos errores se pueden dar por muchos factores y no solo por amenazas de seguridad, ya que un simple cable roto puede suponer un retraso en el acceso de la informacion, ya que se asume que los multiples protocolos implementados mantiene segura la red en la que estamos, y los errores presentados en su mayoria deben ser de indole fisica



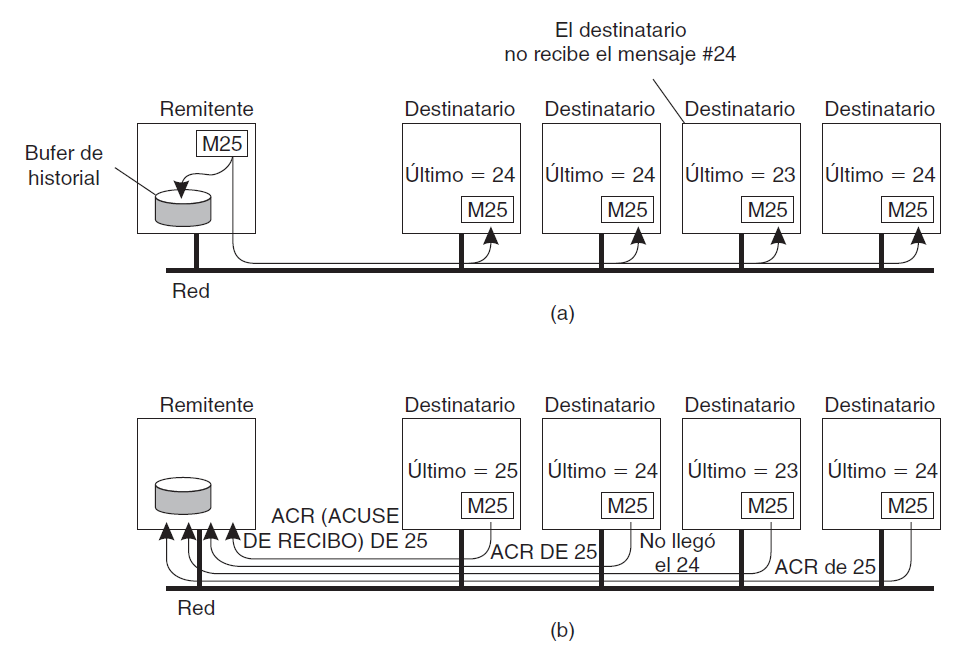
## **3.4 Comunicación confiable en grupo**

Aunque la mayoría de las capas de transporte que nos ofrecen unos canales confiables para la comunicación de punto a punto, ya que es rara la vez nos ofrecen una comunicación confiable a un conjunto de ciertos procesos. Su mejor virtud es el poder permitir que cada uno de los procesos establezca una conexión de punto a punto con cualquier otro proceso con el que desee comunicarse.

Debemos saber distinguir entre una comunicación que es confiable en presencia de ciertos procesos que pueden estar defectuosos y comunicación confiable que en la multitransmisión que es confiable cuando nos puede garantizar que todos los miembros no defectuosos del recibirán un mensaje.

Cuando una situación se simplifica cuando suponemos que existe un acuerdo previo sobre un miembro del grupo y quien no lo es. En particular, al suponer que ciertos procesos no llegaran a fallar, y que no se unen al grupo o no los dejan mientras la comunicación se está llevando a cabo, multitransmisión confiable significa que simplemente que todo mensaje deberá ser entregado a cada miembro actual del grupo. En unos de los casos más simples, no existe ningún requerimiento de que todos los miembros del grupo reciban mensaje en el mismo orden, aunque en algunas de las ocasiones se requiere cumplir con esta característica.

Esta es una forma más débil de multitransmisión confiable es relativamente fácil de poder implementar, ya que está sujeta de nuevo a una condición de que el número de destinatarios este limitado. Pero ahora consideremos el caso en que un solo remitente que desea poder enviar un mensaje a varios destinatarios. Ahora supongamos que el sistema de comunicación que se está empleando una multitransmisión ofrece solo una multitransmisión confiable, lo que cual significa que un mensaje multitransmitido ofrece solo multitransmisión confiable, lo cual significa que el mensaje que es multitransmitido puede perderse alguna de sus partes y llegar a ser entregado de esta forma, pero no le llegaría a todos los remitentes que se tenían complementados.



El proceso de remitente asigna a un numero de secuencia a cada uno de los mensajes que se llegan a multitransmitar. Con esto se supone que los mensajes que se reciben en el orden con el que fueron enviados cada uno. Así es que, de este modo, es mucho más fácil que un destinatario pueda detectar de que no recibió un mensaje. Cada uno de los mensajes multransmitido se guarda localmente en un búfer de historial en el remitente.

Supongamos que el remitente conoce a sus destinatarios, esto simplemente conserva el mensaje en su búfer de historial hasta que cada uno de los destinatarios haya podido devolver un acuse de recibo. Pero si un destinatario detecta que se perdió un mensaje, regresa un acuse de recibo negativo, solicitando al remitente que vuelva a retrasmitir el mensaje. Si no de otro modo, el remitente puede retrasmitir automáticamente el mensaje cuando no se haya recibido por todos los acuses de recibo dentro de cierto tiempo.

Escalabilidad en multitransmisión confiable

Uno de los problemas principales con el esquema de multitransmisión confiable que se acaba de describir es que no puede soportar un gran número de destinatarios. Pero si existen ***N*** destinatarios, el remitente debe estar preparado para aceptar por lo menos ***N*** acuses de recibo. Con muchos destinatarios, el remitente puede verse abrumado por los mensajes de retroalimentación, ello también se conoce como implosión de retroalimentación. Debemos tomar en cuenta que también se requiere tomar en cuenta que los destinatarios se encuentran esparcidos por toda la red de un área amplia.

Una de las soluciones a estos problemas es no hacer que los destinatarios confirmen la recepción de un mensaje. Pero en cambio un destinatario devuelve el mensaje de retroalimentación solo para informar que el remitente no ha enviado el mensaje. Así que de esta forma solo regresa acuses de recibo negativos que generalmente puede llegar a trabajar a mayor escala, aunque no se pueden dar garantías escritas de que nunca ocurrirán implosiones de retroalimentación.

Otro de los problemas es con la devolución de solo acuses de recibo negativos es que el remitente, en teoría, se verá obligado a conservar por siempre un mensaje en un bufer de historial. Como puede ser que el remitente nunca sepa si es que un mensaje ha sido entregado correctamente a todos los destinatarios, siempre este deberá estar preparado por si un destinatario le solicita la retrasmisión de un mensaje viejo.

El remitente borrara un mensaje de su búfer de historial después de transcurrido cierto tiempo y todo esto para poder a llegar a sobrecargar el búfer.

Para multitransmision confiable es reducir el número de mensajes de retroalimentación devueltos al remitente. Un modelo popular usado en varias aplicaciones de área amplia es la supresión de retroalimentación. Este esquema sirve como un fundamento al protocolo de multitransmisión confiable escalable que fue desarrollado pro Floyd y colaboradores.

En SRM los destinatarios nunca confirman una entrega exitosa de un mensaje multitransmitido, sino más bien que informan solo cuando no reciben un el mensaje correspondiente. El cómo detectar la pérdida de un mensaje se deja a la aplicación. Solo se devuelven acuses de recibo negativos como retroalimentación.

La multitransmisión mediante retroalimentación permite que otro miembro del grupo cancele su propia retroalimentación. Supongamos que varios destinatarios no reciben el mensaje *m*. Cada

quien tendrá que devolver un acuse negativo al remitente, *S*, de modo que *m* pueda ser retransmitido.

No obstante, si se supone que las retransmisiones siempre se multitransmiten a todo el grupo,

es suficiente con que *S* reciba sólo una solicitud de retransmisión.

Por esta razón, un remitente R que no recibió un mensaje m programa un mensaje de retroalimentación

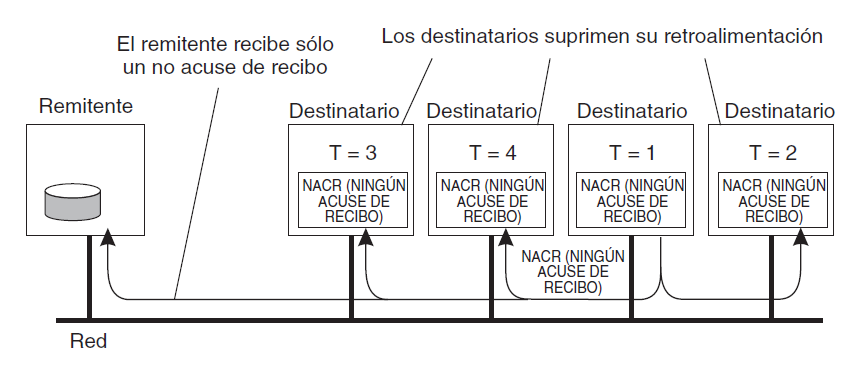
con cierta demora aleatoria. Es decir, no envía la solicitud de retransmisión sino hasta

que haya transcurrido cierto tiempo aleatorio. Si, en el ínterin, R recibe otra solicitud de retransmisión

de m, R cancelará su propia retroalimentación, a sabiendas de que m será retransmitido en breve.

De esta manera, idealmente, S recibirá sólo un mensaje de retroalimentación, el que a su vez

retransmite luego m.



La supresión de retroalimentación ha mostrado escalar razonablemente bien, y ha sido utilizada

como el mecanismo fundamental en varias aplicaciones colaborativas en internet tal como un

pizarrón compartido (Shared whiteboard) compartido. No obstante, este método también presenta

varios problemas serios. En primer lugar, garantizar que el remitente reciba sólo una solicitud de

retransmisión requiere una programación razonablemente precisa de los mensajes de retroalimentación por parte de cada destinatario. De lo contrario, muchos destinatarios seguirán enviando su retroalimentación al mismo tiempo. No es tan fácil ajustar los cronómetros como corresponde en un grupo de procesos dispersos a través de una red de área amplia.

Otro problema es que la multitransmisión de la retroalimentación también interrumpe aquellos

procesos que recibieron el mensaje exitosamente. En otros términos, otros destinatarios se ven obligados a recibir y procesar mensajes inútiles para ellos. La única solución a este problema es permitir que los destinatarios que no hayan recibido el mensaje m se unan a un grupo de multitransmisión distinto para m, como se explica en Kasera y colaboradores

Para simplificar las cosas, supongamos que existe sólo un remitente que debe transmitir mensajes

a un grupo muy grande de destinatarios. El grupo de destinatarios se divide en varios subgrupos,

ello posteriormente se organiza en un árbol. El subgrupo que contiene la remitente forma la

raíz de su árbol. Dentro de cada subgrupo, se puede utilizar cualquier esquema de multitransmisión

confiable que funcione para grupos pequeños.

Cada subgrupo designa un coordinador local, el cual es responsable de manejar las peticiones

de retransmisión de los destinatarios contenidos en su subgrupo. El coordinador local dispondrá,

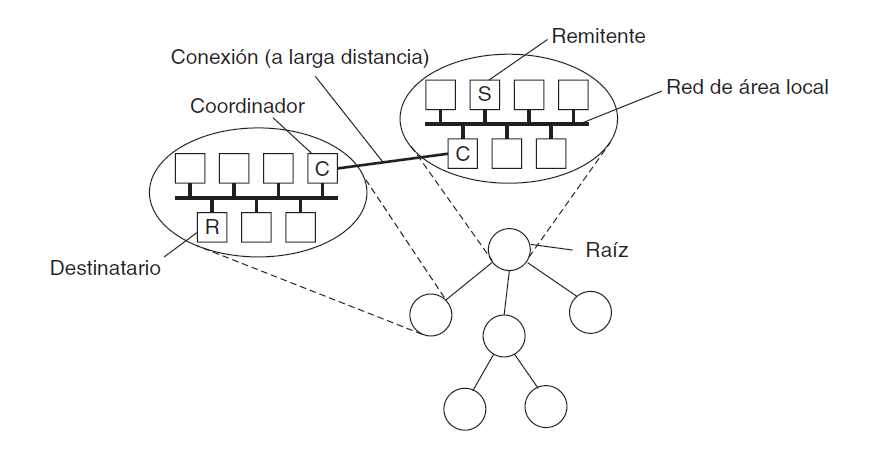
por lo tanto, de su propio búfer de historial. Si el coordinador no envía un mensaje m, le pide al

coordinador del subgrupo padre que lo retransmita. En un esquema de basado en acuses de recibo,

un coordinador local envía un acuse de recibo a su padre si recibió el mensaje. Si un coordinador

recibe acuses de recibo del mensaje m de todos los miembros que integran su subgrupo, y también

de su hijo, puede eliminar m de su bufer de historial.



Para ver por qué es tan importante la atomicidad, consideremos una base de datos replicada

construida como una aplicación encima de un sistema distribuido. El sistema distribuido ofrece servicios de multitransmisión confiables. En particular, permite construir grupos de procesos a los que

los mensajes puedan ser enviados con toda confianza. La base de datos replicada se construye por

consiguiente como un grupo de procesos, un proceso por cada réplica. Las operaciones actualizadas

siempre son multitransmitidas a todas las réplicas y posteriormente se realizan a nivel local. En

otros términos, suponemos que se utiliza un protocolo de replicación activa.

Supongamos que ahora se debe realizar una serie de actualizaciones, pero que durante la ejecución

de una de ellas se congela una réplica. Por consiguiente, la actualización de dicha réplica se

pierde, aunque, por otra parte, se realiza correctamente en las demás réplicas.

Cuando una réplica recién congelada se recupera, en el mejor de los casos podrá recuperarse al

mismo estado que tenía antes de la congelación; sin embargo, puede que se hayan perdido varias

actualizaciones. En ese punto, resulta esencial que se ponga al día con las demás réplicas. Llevar la

réplica al mismo estado de las demás requiere saber con exactitud qué operaciones se perdió y en

qué orden se deben realizar éstas.

## **3.5 Recuperación**

La recuperación de errores es fundamental para la tolerancia a fallas. Recordemos que un error es

esa parte de un sistema que puede conducir a una falla. La idea integral sobre recuperación de errores

es reemplazar un estado erróneo con un estado libre de error. Esencialmente, existen dos formas

de recuperación de errores.

En la recuperación hacia atrás, lo principal es hacer que el sistema regrese de su estado

actual erróneo a su estado previamente correcto. Para lograrlo, será necesario registrar el estado

del sistema de vez en cuando y, cuando las cosas vayan mal, restaurar el estado registrado.

Cada vez que se registra (una parte de) el estado actual del sistema, se dice que se marca un

punto de control.

Otra forma de recuperación de errores es la recuperación hacia adelante. En este caso, cuando

el sistema ha entrado a un estado erróneo, en lugar de regresarlo a un estado de punto de control

previo, se intenta llevarlo a un nuevo estado correcto a partir del cual se pueda continuar ejecutando.

El problema principal con los mecanismos de recuperación de errores hacia adelante es que se

debe saber de antemano qué errores pueden ocurrir. Sólo en ese caso es posible corregir los errores

y trasladarse a un nuevo estado.

La distinción entre la recuperación de errores hacia atrás y hacia adelante se explica con facilidad

cuando consideramos la implementación de comunicación confiable. El método común de recuperar

un paquete perdido es permitir que el remitente retransmita el paquete. En efecto, la

retransmisión de paquetes establece que intentamos regresar a un estado correcto previo, es decir,

a aquél en el que se perdió el paquete que está siendo enviado. La comunicación confiable a través

de la retransmisión de paquetes es, por consiguiente, un ejemplo de la aplicación de técnicas de recuperación de errores hacia atrás.

Un procedimiento alternativo es utilizar un método conocido como corrección por borradura.

En este procedimiento, se construye un paquete perdido a partir de otro para entregar los paquetes

con éxito. Por ejemplo, en un código de borradura de bloque (n,k), un conjunto de k paquetes

originales se cifra en la forma de n paquetes cifrados, de modo que basta un juego de k paquetes cifrados para reconstruir los k paquetes originales. Valores típicos son k = 16 o k = 32, y k = n = 2k. Si aún no se han entregado suficientes paquetes, el remitente deberá continuar transmitiendo paquetes hasta que un paquete previamente perdido pueda ser construido. La corrección por borradura es un ejemplo típico de un método de recuperación de errores hacia adelante.

Almacenamiento estable

Para recuperarse a un estado previo, es necesario que la información requerida para habilitar la recuperación sea guardada con seguridad. Seguridad en este contexto significa que la recuperación

sobreviva a congelaciones de proceso y fallas de sitio, pero quizá también a varias fallas de medios

de almacenamiento. El almacenamiento estable desempeña un rol muy importante cuando se trata de

recuperación en sistemas distribuidos.

El almacenamiento se presenta en tres categorías. Primero, existe una memoria RAM ordinaria

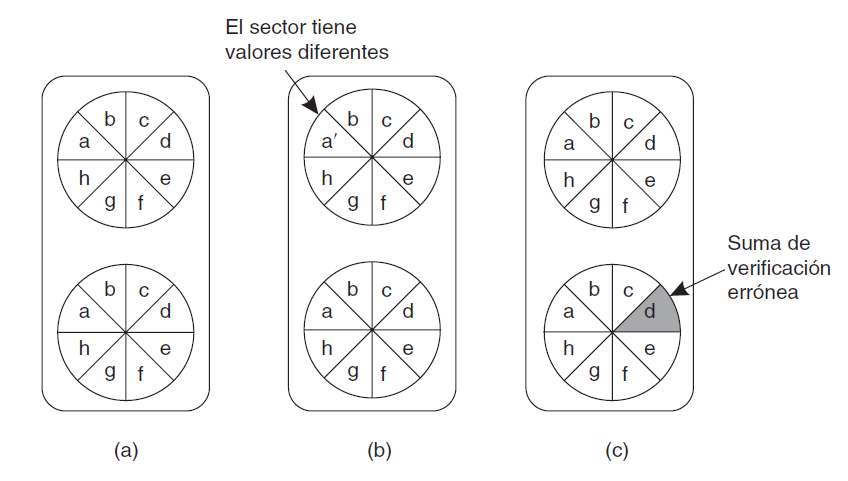
que se borra cuando falla la corriente o una máquina se congela. En segundo lugar tenemos un

almacenamiento en disco, el cual sobrevive a fallas de la CPU pero también se puede perder cuando

ocurren fallas de cabeza de disco.

Por último, también existe **almacenamiento estable**, el cual está diseñado para sobrevivir a

cualquier cosa excepto a calamidades extremas tales como inundaciones o terremotos. El almacenamiento estable puede ser implementado con un par de discos ordinarios. Cada bloque grabado en el disco 2 es una copia exacta del bloque correspondiente incluido en el disco 1. Cuando un bloque se actualiza, primero se actualiza y verifica el bloque en el disco 1, luego se hace lo mismo en el correspondiente bloque del disco 2.



Marcación de puntos de control

En un sistema distribuido tolerante a fallas, la recuperación de errores hacia atrás requiere que el

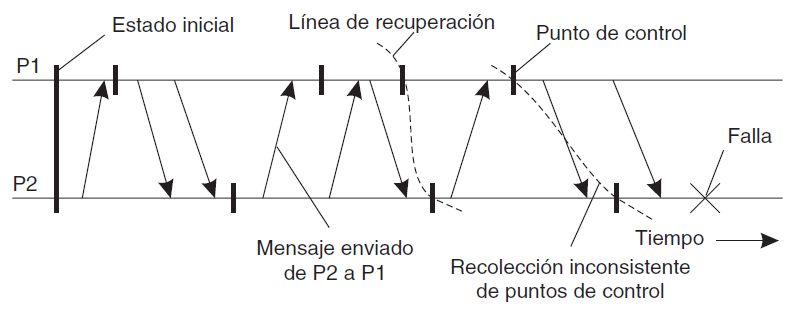
sistema guarde con regularidad su estado en almacenamiento estable. En particular, se tiene que

registrar un estado global consistente, llamado también instantánea distribuida. En una instantánea

distribuida, si un proceso P registró la recepción de un mensaje, entonces también deberá haber un

proceso Q que registró el envío de dicho mensaje. Después de todo, el mensaje debe provenir de

alguna parte.



En esquemas de recuperación de errores hacia atrás, de vez en cuando cada proceso guarda su

estado en un almacenamiento estable localmente disponible. Para recuperarse después de una falla

de proceso o de sistema, se requiere construir un estado global consistente a partir de esos estados

locales. En particular, es mejor recuperarse a la instantánea distribuida más reciente, también conocida

como línea de recuperación. En otros términos, una línea de recuperación corresponde a la

recolección consistente más reciente de puntos de control.

Marcación de puntos de control independiente

Por desgracia, la naturaleza distribuida de la marcación de puntos de control (en la que cada proceso

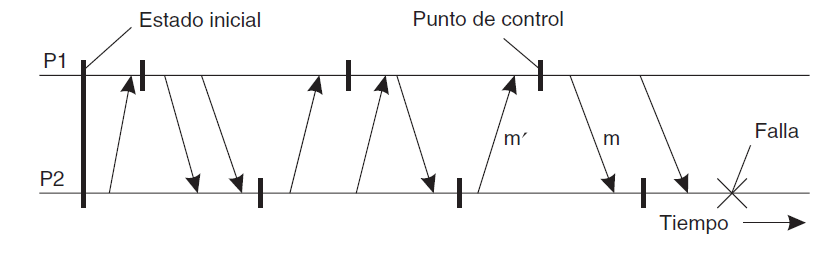
simplemente registra su estado local de vez en cuando de una manera no coordinada) puede hacer

difícil encontrar una línea de recuperación. Para descubrir una línea de recuperación se requiere

que cada proceso retroceda a su estado más recientemente guardado. Si estos estados locales juntos

no conforman una instantánea distribuida, es necesario más retroceso.

Este proceso de un retroceso en cascada puede conducir a lo que se conoce como efecto dominó.



Cuando un proceso P2 se congela, se tiene que recuperar su estado al punto de control más

recientemente guardado. En consecuencia, el proceso P1 también tendrá que ser retrocedido. Por

desgracia, los dos estados locales más recientemente guardados no forman un estado global consistente: el estado guardado por P2 indica la recepción de un mensaje m, pero ningún otro proceso puede ser identificado como su remitente. Por tanto, P2 tiene que ser retrocedido hasta un estado previo. Sin embargo, el siguiente estado al cual P2 es retrocedido tampoco puede ser utilizado como

parte de una instantánea distribuida.

En este caso, P1 habrá registrado la recepción del mensaje m´, pero no existe ningún evento registrado de que este mensaje esté siendo enviado. Por consiguiente, es necesario retroceder también P1 a un estado previo. En este ejemplo, resulta que la línea de recuperación es en realidad el estado inicial del sistema.

Como los procesos marcan puntos de control locales independientes entre sí, este método también

se conoce como marcación de puntos de control independiente. Una solución alternativa

es coordinar globalmente la marcación de puntos de control, como veremos a continuación, pero

la coordinación requiere sincronización global, la cual puede presentar problemas de desempeño.

Otra desventaja de la marcación de puntos de control independiente es que cada almacenamiento

local tiene que ser limpiado periódicamente, por ejemplo, haciendo funcionar un recolector de basura

distribuido especial. Sin embargo, la desventaja principal radica en el cálculo de la línea de

recuperación.

Marcación de puntos de control coordinada

Como su nombre lo sugiere, en la marcación de puntos de control coordinada todos los procesos

se sincronizan para escribir conjuntamente su estado en un almacenamiento local estable. La

ventaja principal de la marcación de puntos de control coordinada es que el estado guardado es

automática y globalmente consistente, de modo que se evite que los retrocesos conduzcan al efecto

dominó. El algoritmo de instantánea distribuido que presentamos en el capítulo 6 puede ser utilizado

para coordinar la marcación de puntos de control coordinada. Este algoritmo es un ejemplo de coordinación de marcación de puntos de control que no se bloquea.

Una solución más simple es utilizar un protocolo de bloqueo bifásico. Un coordinador primero

multitransmite un mensaje CHECKPOINT\_REQUEST a todos los procesos. Cuando un proceso

recibe un mensaje como ese, toma un punto de control local, pone en cola cualquier mensaje subsiguiente entregado a él por la aplicación que se está ejecutando, y confirma al coordinador que ha tomado un punto de control. Cuando el coordinador recibe una confirmación de todos los procesos,

multitransmite un mensaje CHECKPOINT\_DONE para permitir que los procesos (bloqueados)

continúen.

# **Conclusiones**

En esta unidad vimos lo correspondiente a las tolerancias a fallas, de este modo comprendimos todos los mecanismos y protocolos que tienen que ser aplicados a un sistema para que este se pueda considerar funcional, del mismo modo, como es que estos protocolos son evaluados, y definidos, no basta con que mantengan una conexión, o nos permita recuperar un dato, si no en una serie de diversos factores que nos permitan trabajar tanto en el lugar que residimos como en lugares remotos los cuales pueden estar muy alejados y es necesario que cumplan con ciertas características para que podamos decir que el sistema puede tolerar fallas, no hablamos solo de desconexiones, si no, también de protocolos de protección de múltiples y diversos ambientes en los cuales se puede mover información muy delicada, nos vemos obligados a evitar fallos mínimos como el desface de los relojes internos de los ordenadores, y con ello el error que este conlleva, al igual que la espera muy larga por una respuesta de un dato que bajemos de la web, esto muchas veces hecho para proteger la integridad de estos sistemas distribuidos, generando una visión mas clara sobre lo que conlleva trabajar previendo las fallas de los sistemas

# **Herramientas y recursos**

* Imágenes de Google
* Power Point para crear la exposición
* Youtube
* El documento de nuestra investigación del tema
* Scribbr para el APA
* Equipo de cómputo.

# **Bibliografía**

Sistemas Operativos II - Gestión de Curso Edgard. (s. f.). <https://sites.google.com/site/gestiondecursoedgard/agosto---diciembre-2018/sistemas-operativos-ii>

Sistemas Distribuidos - Tolerancia a fallos. (2018, 8 enero). <https://programacion-js.blogspot.com/2018/01/sistemas-distribuidos-tolerancia-fallos.html>

*2.1 Comunicación: comunicación con cliente – servidor, comunicación con llamada a procedimiento remoto, comunicación en grupo, tolerancia a fallos. - mrTripas*. (s. f.). https://sites.google.com/site/mrtripus/home/sistemas-operativos-2/2-1-comunicacion-comunicacion-con-cliente-servidor-comunicacion-con-llamada-a-procedimiento-remoto-comunicacion-en-grupo-tolerancia-a-fallos

*404 Not Found*. (s. f.). http://cs.uns.edu.ar/%7Esd/data/apuntes/SD-2019-mod+03.pdf

*Modelo de Comunicación de Redes Informaticas*. (2022, 13 junio). MindMeister. https://www.mindmeister.com/es/839898047/modelo-de-comunicaci-n-de-redes-informaticas