El Protocolo de Membresía Grupal de Tres Ruedas como Base de un Sistema Distribuido Tolerante a Fallas.

Guillermo R. Friedrich [1] *y Jorge R. Ardenghi* [2]

(1) Depto. Electrónica - Univ. Tecnológica Nac. – Fac.Reg.B.Blanca
(2) Depto. Cs. de la Computación – Univ. Nac. del Sur
gfried@frbb.utn.edu.ar; jra@cs.uns.edu.ar

Resumen

Una de las técnicas más usuales para lograr la tolerancia a fallas en sistemas distribuidos consiste en utilizar servicios replicados. Si bien existen diversas técnicas de replicación: pasiva (primario/backup), activa (p.ej.: triple redundancia modular) y semi-activa (líder/seguidores); sin embargo todas ellas requieren disponer de un grupo de procesadores sobre los cuales ejecutan las réplicas de dichos servicios. Por otra parte, dependiendo del rol del tiempo en estos sistemas se puede hacer una distinción entre el modelo sincrónico, el modelo asincrónico, y el más recientemente introducido modelo asincrónico temporizado [1]. El presente trabajo describe brevemente este último modelo, y posteriormente trata de la aplicación del protocolo de membresía de tres ruedas con grupos mayoritarios [2] para la conformación de un grupo de procesadores con capacidad de reconfigurarse ante caídas, fallas de comunicación y/o recuperación de aquellas. Finalmente se describe y evalúa una implementación de este protocolo, basada en sockets, sobre una red de computadoras personales.

1. Introducción

Las diferentes técnicas de replicación [3][4] que pueden utilizarse para implementar servicios tolerantes a fallas requieren de primitivas de multienvío para su operación. La abstracción de la *comunicación grupal* es un marco adecuado para proveer de tales primitivas [5][6][7]. Se puede considerar a la comunicación grupal como la infraestructura sobre la que se basan las técnicas de replicación, tal como se muestra en la siguiente figura:

Capas Superiores
Técnicas de Replicación
Comunicación Grupal
Envío y Recepción de Datagramas
Sistema Operativo

A fin de implementar servicios distribuidos con alta disponibilidad, los *procesadores* son organizados en *equipos*. Un equipo de procesadores se define como el conjunto de todos los procesadores capaces de implementar un determinado servicio.

En principio, hay dos aproximaciones para manejar la comunicación entre miembros de un equipo que implementan un servicio distribuido. En la primera de ellas, la aproximación *independiente de la historia*, cuando un miembro *p* del equipo debe responder a una petición de servicio que requiere que se comunique con los demás miembros, *p* intenta comunicarse con todos ellos sin tener en cuenta si en el pasado reciente ha podido o no hacerlo satisfactoriamente. Un ejemplo de esta aproximación podría ser un servicio replicado basado en la votación, con tres procesadores *p, q* y *r*; en este caso, cuando *p* debe realizar una actualización intenta comunicarse con *q* y *r*, sin tener en cuenta si en los últimos intentos tuvo éxito o no.
En la otra aproximación, *dependiente de la historia*, los miembros del equipo se adaptan a fin de comunicarse solamente con aquel *grupo* de miembros con los que hayan tenido una comunicación exitosa en los últimos tiempos. La idea es permitir a los miembros del equipo que tienen una historia reciente de

comunicaciones exitosas, que formen dinámicamente grupos de comunicación y trabajo. Cuando se pierde la comunicación con un procesador este es eliminado del grupo, mientras que cuando un procesador se restablece, el mismo es admitido en un nuevo grupo.

En general, la implementación de servicios distribuidos basada en grupos de comunicación se caracteriza por un menor tráfico de mensajes y mejor tiempo de respuesta. La principal complicación introducida por la aproximación dependiente de la historia es la necesidad de contar con un servicio de *membresía grupal*.

Otro servicio utilizado en la comunicación grupal que también tiene la finalidad de *simplificar la programación* de sistemas replicados es la *difusión atómica* [8][9][11].

La *membresía* brinda acuerdo sobre los grupos de servidores que prestan un servicio a lo largo del tiempo, mientras que la *difusión atómica* brinda acuerdo acerca de la historia de las actualizaciones de estados realizadas en tales grupos.

También cabe hacer una distinción entre dos modelos de servicios de comunicación grupal: sincrónicos y asincrónicos [10]. Los primeros son aptos para sistemas de alta disponibilidad con requerimientos de tiempo real duro (hard real-time), mientras que los segundos son aptos cuando los requerimientos de tiempo real son moderados (soft real-time).

En el siguiente punto se describe el modelo asincrónico temporizado [1], con una breve referencia al modelo sincrónico. Cabe destacar que se han tenido las siguientes consideraciones para simplificar el análisis:

- Se considera un único servicio S implementado por servidores replicados sobre un conjunto fijo de procesadores P, y se considera que hay una correspondencia uno a uno entre servidores y procesadores.
- Del total de procesadores que conforman el equipo P, algunos integran el grupo de servidores G, que en un intervalo de tiempo dado implementan el servicio S. Los motivos por los cuales algunos procesadores de P no forman parte de G pueden ser o bien porque han sufrido una falla, o bien porque han salido de servicio voluntariamente.

2. El Modelo Asincrónico Temporizado.

Se considera que hay un grupo de servidores ubicados cada uno en diferentes procesadores, y que la comunicación entre estos procesos se realiza únicamente a través de mensajes (datagramas). Si bien los mensajes pueden perderse o demorarse indefinidamente, la mayoría llegarán a destino dentro de un cierto tiempo máximo conocido *d*. Por lo tanto, la comunicación mediante datagramas presenta una semántica de falla del tipo de *omisión/performance*.
Los servidores son planificados para ejecutarse en respuesta a eventos tales como la llegada de mensajes y los timeouts. A pesar de que los retardos de planificación no estén acotados, en *muchos* casos estos retardos son menores que un cierto valor constante *s*. Cuando los retardos de planificación exceden *s*, los servidores ven afectada su performance. Como los procesadores y los servidores usan mecanismos de auto-verificación, es poco probable que produzcan resultados funcionalmente erróneos. Por lo tanto, los servidores presentan una semántica de falla de *caída/performance*. Los valores topes *d* y *s* mencionados determinan que el peor caso de retardo en la comunicación servidor-servidor es $d = s + d + s$. En [2] y [10] se define a un sistema que cumple con las hipótesis recién descriptas como *sistema asincrónico temporizado (timed asynchronous)*. La introducción de $d + s$ hace que la especificación de los servicios del procesador y de comunicación incluyan una referencia al tiempo, es decir que no sólo deben especificarse las transiciones de estados y las salidas que deben producirse ante una cierta entrada, sino también el intervalo de tiempo real dentro del cual se espera que esto ocurra.

En contraposición, un sistema asincrónico sin restricciones temporales (*time-free*) es aquel en el que sólo se define cual debe ser el próximo estado/salida ante una dada entrada. Se considera que su funcionamiento es "correcto" aunque tarde años en responder a una entrada. En consecuencia, esta definición de "correctitud" impide la implementación de servicios tolerantes a fallas, tales como el

consenso y la membresía. Dado que, en la práctica, se requiere que los sistemas sean tolerantes a las fallas, es natural que los mismos sean temporizados y que hagan uso de timeouts.

Los sistemas asincrónicos permiten que la mayor parte del tiempo la comunicación esté acotada en el tiempo. Sin embargo, debido a la congestión u otros fenómenos adversos, los procesadores pueden estar temporariamente desconectados. Se pueden hacer las siguientes definiciones:

- Dos procesadores *p* y *q* están *conectados* durante un intervalo de tiempo [*t, t'*] si ambos están correctos (no caídos y en tiempo), y cada mensaje enviado en el intervalo [*t, t'-d*] es entregado dentro de un tiempo *d*.

- Dos procesadores *p* y *q* están *desconectados* durante un intervalo de tiempo [*t, t'*] si ningún mensaje enviado entre ellos en el intervalo [*t, t'*] llega a destino, o si *p* o *q* están caídos durante [*t, t'*].

- Dos procesadores *p* y *q* están *parcialmente conectados* durante el intervalo [*t, t'*] si no están ni *conectados* ni *desconectados* durante [*t, t'*] . Por ejemplo, la sobrecarga transitoria de la red de comunicación puede hacer que algunos mensajes entre *p* y *q* se pierdan o se demoren excesivamente.

- Un sistema asincrónico es estable en [*t, t'*] si durante [*t, t'*] se cumple que:
    - Ningún procesador falla o se restaura.
    - Todos los pares de procesadores están conectados o desconectados
    - La relación "conectado" entre procesadores es transitiva.

Debido a la baja tasa de fallas que presentan los procesadores y sistemas de comunicación actuales, los sistemas asincrónicos bien "sintonizados" pueden alternar entre períodos muy largos de estabilidad y períodos relativamente cortos de inestabilidad.

3. Comunicación Grupal Asincrónica

La comunicación grupal sincrónica simplifica considerablemente la programación de aplicaciones replicadas, ya que asegura que cada réplica tenga un conocimiento actualizado del estado del sistema; sin embargo tiene un costo importante, y es que se cumplan en tiempo real las siguientes hipótesis [10]:

H1. Todos los retardos de comunicación son menores que δ; todos los retardos de planificación de los procesos que implementan la difusión y el broadcast son menores que *s*.
H2. La cantidad de componentes de comunicación, tales como procesadores y enlaces, que pueden estar fallando durante cualquier difusión está acotada a un valor constante F.
H3. La red tiene suficientes caminos redundantes entre dos procesadores cualesquiera *p* y *q*, tal que *q* siempre recibirá una copia de cualquier mensaje difundido por *p*, aunque haya hasta F componentes fallando.
H4. La tasa a la cual se inician las difusiones es limitada mediante métodos de control de flujo; esta tasa es menor que la tasa máxima a la cual los procesadores y los servidores pueden recibir y procesar los mensajes difundidos.

Las metas a alcanzar por los servicios de comunicación grupal asincrónica pueden ser similares a las previstas para el caso sincrónico, es decir:

G1. Lograr el acuerdo sobre la historia lineal del grupo de servidores; y
G2. Para cada grupo *g*:
    a) acuerdo sobre el estado inicial de *g*, y sobre la historia lineal de actualizaciones de *g*
    b) para grupos sucesivos *g, g'*, asegurar que todos los miembros de *g'* hereden correctamente el estado replicado mantenido por los miembros de *g*.

Sin embargo, la incertidumbre en la comunicación introduce una serie de complicaciones:

- Como los procesos no pueden distinguir entre fallas de procesos o de comunicaciones, a fin de obtener acuerdo sobre la historia lineal de los grupos, se deben imponer algunas restricciones. Por ejemplo, se puede restringir de tal modo que solamente los *grupos mayoritarios* contribuyan a la historia (un *grupo mayoritario* es aquel formado por una mayoría numérica sobre el total de miembros del equipo, es decir $|members(g)| > |P| / 2$).
  Esta restricción no sólo permite obtener G1, sino que también permite asegurar G2b ya que dos grupos mayoritarios sucesivos deben tener al menos un miembro en común para que se cumpla tal restricción.

- Mientras que es posible diseñar protocolos asincrónicos con similares propiedades de seguridad que los sincrónicos, las propiedades temporales son mucho más livianas; los retardos con los cuales los procesos aprenden sobre nuevas actualizaciones, uniones y fallas están acotados en el tiempo sólo cuando se mantienen ciertas condiciones de estabilidad.

- Debido a que en los sistemas asincrónicos los retardos no están acotados, asegurar el acuerdo sobre el estado inicial requiere más trabajo que en los sistemas sincrónicos.

3.1 Protocolos de Membresía para Grupos Asincrónicos.

Cristian y Schmuck [2] han analizado y propuesto cinco protocolos, de complejidad creciente, para el manejo de la membresía en grupos asincrónicos temporizados. De estos cinco, el que presenta la mejor relación costo/beneficio es el protocolo de membresía grupal de tres ruedas para grupos mayoritarios completos.

Puede pensarse que un protocolo para el manejo de la membresía grupal tiene dos componentes:
- un protocolo para la detección de cambios en el sistema (por ej.: procesadores que caen y procesadores que se reinician), y,
- un protocolo para informar estos cambios, de manera consistente, a los procesadores correctos.

Los diferentes protocolos de membresía tratados en [2] difieren en el segundo de estos componentes, es decir: como actualizan la información a los procesadores correctos; sin embargo todos ellos utilizan los mismos mecanismos para detectar los cambios en el sistema.

3.1.1 Detección de Caídas de Procesadores.

Una complicación propia de los sistemas distribuidos asincrónicos temporizados es la imposibilidad que tiene un procesador p que no logra comunicarse con otro procesador q, dentro de un cierto tiempo $2d$, para distinguir entre las siguientes situaciones posibles:

- El procesador q ha caído.
- El procesador q está lento.
- Se ha caído el subsistema de comunicación.
- Hay una falla transitoria del subsistema de comunicación (tal vez una sobrecarga) que hace que se pierdan o se demoren demasiado uno o varios mensajes entre p y q.

Como uno de los objetivos de utilizar grupos de procesadores es enmascarar las caídas de procesadores, cuando p no puede comunicarse con q, p considera que q ha fallado. Es decir: p decide que q ha caído si esperaba recibir un mensaje de q dentro de un cierto intervalo de tiempo, y finalizado el mismo no lo recibió.

Un mecanismo eficiente y con gran economía de mensajes, que permite detectar la salida de un procesador, es mediante el siguiente algoritmo basado en una *lista de asistentes*:

Se asume que el sistema es actualmente estable y se considera una partición física S con más de un procesador. Se puede definir un orden cíclico en función de los identificadores de cada procesador; por ejemplo: los vecinos izquierdo y derecho de *p* serán aquellos procesadores con el identificador anterior y posterior dentro de S (un caso especial se da con los que tienen el menor y el mayor identificador, los que serán vecinos entre sí).
Uno de los procesadores (por ejemplo el de menor identificador) pone a circular un mensaje (*"estoy-vivo"*) hacia la "derecha" cada π unidades de tiempo. Si el mensaje es retransmitido por todos los procesadores en S y llega a quien lo originó dentro de un cierto tiempo, entonces todos los procesadores en S saben que ninguno ha fallado. Si cualquier procesador detecta que el mensaje se ha perdido o circula lentamente, asume que uno de ellos puede haber fallado. Este método de detección requiere de sólo *n* mensajes cada π unidades de tiempo.

3.1.2 Detección de Procesadores que se recuperan.

La detección de un procesador que se recupera es bastante sencilla. Un procesador que se reinicia luego de una caída puede enviar mensajes de "sondeo" a todos los demás procesadores para hacerles conocer tal evento. Sin embargo, debido a que algunos mensajes se pueden perder ya sea por fallas transitorias de comunicación, o bien porque los procesadores pueden estar desconectados por un tiempo relativamente largo, estos mensajes de "sondeo" se deben enviar periódicamente, cada μ unidades de tiempo. A fin de reducir la cantidad de mensajes, sólo los líderes de grupos minoritarios envían mensajes de sondeo. Cuando todos los procesadores sean miembros de un único grupo mayoritario no se enviarán mensajes de este tipo.

4. El protocolo de membresía grupal de tres ruedas con detección de partición.

4.1 Consideraciones previas y definiciones.

Debido a que en los sistemas asincrónicos el retardo que sufren los mensajes es aleatorio, que las fallas de comunicación y de los procesadores ocurren en momentos impredecibles, y que los relojes pueden desviarse del tiempo real a tasas desconocidas para los procesadores, no es realista hablar de que los procesadores que están conectados concuerden en tiempo real acerca de la membresía, ya que la sincronización entre procesadores está excluida de antemano.

Sin embargo, se requiere alguna forma de acuerdo sobre cuales procesadores están levantados y conectados y cuales no lo están. Un ejemplo concreto de esta necesidad se puede ver a continuación:

> si *p* y *q* son dos servidores encargados de manejar la política de disponibilidad primario/backup para un grupo de dos administradores de base de datos *{d, d'}* situados en los mismos procesadores que *p* y *q* respectivamente. Si luego de un período de estabilidad durante el cual *d* cumplió el papel de primario, se pierde un mensaje "estoy-vivo" enviado por *p*, entonces *q* detectará eso como una falla de *p* y promoverá a *d'* al rango de primario; esto producirá una violación a una de las condiciones de seguridad de la política de primario/backup que dice que en ningún instante, en tiempo real, podrá haber más de un primario.
> Este comportamiento indeseado se denomina de "cabeza dividida", y es lo que se intenta solucionar mediante los protocolos de membresía.

Se introduce la noción de *grupo de procesadores*, en el que se reunen aquellos procesadores activos y que en el pasado reciente han podido comunicarse satisfactoriamente, y que concuerdan en comunicarse únicamente con los demás miembros de su grupo.

En lugar de buscar el objetivo imposible de estar de acuerdo en tiempo real, sólo se requiere que los procesadores unidos a un mismo grupo coincidan en la identidad de los miembros de tal grupo (a pesar de que esto suceda en tiempo real diferente). En cualquier instante, un procesador sólo puede estar unido a

lo sumo a un único grupo. Como a lo largo del tiempo se van creando nuevos grupos a medida que los procesadores se desconectan o se reconectan, se requiere que un protocolo de membresía grupal permita identificar todos los grupos mediante un *identificador de grupo de procesadores* g. Se denomina G al conjunto de todos los posibles identificadores de grupo provistos por el protocolo. Se requiere que el protocolo de membresía grupal establezca un orden total "<" sobre G.

El identificador de grupo es un número que consta de dos partes: la más significativa corresponde al número de secuencia del grupo (n) y la menos significativa corresponde al identificador del procesador que propuso la creación del grupo (p). En función de esto, la relación de orden entre dos identificadores g y g' se establece de la siguiente manera:

$$g < g' \Leftrightarrow (g.n < g'.n) \text{ or } (g.n = g'.n \text{ and } g.p < g'.p)$$

De esta manera se asegura la monotonicidad creciente de los identificadores de grupo, y al mismo tiempo que el identificador de grupo generado por un procesador sea diferente al generado por cualquier otro.

Además, el protocolo utiliza las siguientes variables de estado:

joined: P → {verdadero, falso}	*joined(p)* indica si un procesador *p* está o no unido a un grupo en un instante determinado.
group: P → G	*group(p)* indica (si y sólo si *joined(p)=verdadero*) a que grupo está unido *p* en un instante dado.
members: P → Subconjunto-de-P	*members(p)* indica (si y sólo si *joined(p)=verdadero*) cuales son los procesadores que componen el grupo al que está unido *p* en un instante dado (es decir los procesadores que componen *group(p)*).

Se dice que un procesador "*p* está unido a un grupo *g*", si joined(p)=verdadero y group(p)=g. Asimismo, se dice que "*p* se unió a *g* en el instante t_1" y "*p* abandonó *g* en el instante t_2", si t_1 y t_2 son el primer y el último instante de tiempo real, respectivamente, del intervalo durante el cual *p* estuvo unido a *g*.

Se define como *líder de grupo* al procesador de menor identificador entre los miembros del grupo. El líder de grupo es el encargado de poner a circular el mensaje de "lista de asistentes" cada π unidades de tiempo. Los demás procesadores del grupo monitorean el pasaje de la misma en el tiempo correcto.

Si un procesador es el líder de un grupo minoritario, debe además enviar mensajes de "sondeo" cada μ unidades de tiempo, a los demás procesadores que no pertenezcan a su grupo. Cada mensaje de sondeo incluye la siguiente información:

- el identificador de grupo *g*
- *members(g)*
- *s(r)*, la vista que representa el conocimiento que tiene p de la última estampilla de tiempo enviada por cada procesador $r \in P$.

Un par de propiedades muy importantes, referidas al liderazgo, que se deben cumplir son:

- En cualquier instante debe haber a lo sumo un líder.
- El tiempo que demanda la elección de un nuevo líder no debe ser mayor que un cierto tiempo dado.

Se introduce el concepto de *grupo predecesor*, definido de la siguiente manera:

$$pred(g, p) = \text{máx}\{ g' \mid g' < g \text{ and } p \text{ joined } g' \}.$$

Es decir: para un procesador *p* unido a un grupo *g*, el grupo predecesor es un grupo al que *p* estuvo unido y cuyo identificador *g'* es el mayor entre los menores a *g*.

Si un procesador nunca estuvo unido a un grupo anterior a *g*, se define *pred(g, p) = 0*

Otro concepto que se define es el de *Partición Lógica*. Se dice que dos procesadores *p* y *q* de un grupo *g* estuvieron *particionados lógicamente*, antes de unirse a *g*, si *pred(g,p) ¹ pred(g,q)*. Las particiones lógicas pueden ocurrir en sistemas asincrónicos ya sea por fallas físicas o bien porque algunos mensajes se pierdan o se demoren. Al respecto, el protocolo de membresía debe cumplir con el siguiente requerimiento:

(Sp) *Notificación de Partición Lógica*. Sea *p* un procesador que se une a un grupo *g*. Al momento de unirse un procesador *p* a un grupo *g*, el protocolo de membresía debe hacerle conocer a *p* el valor de *pred(g,q), ∀ q ∈ members(g).*

Cuando dos procesadores detectan que han estado lógicamente particionados, pueden estar en desacuerdo sobre los eventos que han ocurrido desde la última vez que estuvieron en un mismo grupo. Por lo tanto, cuando los procesadores *p* y *q* se vuelven a unir en un grupo *g*, y provienen respectivamente de *g'* y *g"*, con estados *s*' y *s*", deberán acordar en un nuevo estado *s = f(s',s")*, donde *f* es una función de *"mezcla de estados"* [11].
Si las operaciones implementadas por el equipo son conmutativas, como por ejemplo agregar ítems a una lista, la "mezcla de estados" puede consistir simplemente en la unión de ambos estados. Si las operaciones realizadas pueden ser, por ejemplo, insertar o quitar elementos de una lista, la mezcla de estados es más complicada y debe existir alguna regla que permita resolver los conflictos.

4.2 Implementación del protocolo.

En primer lugar, se utiliza una "lista de asistentes" para detectar caídas y recuperaciones de procesadores. Asimismo, se utiliza almacenamiento estable para guardar los identificadores de grupo, los cuales van tomando valores crecientes a medida que se van creando nuevos grupos.
Cuando un procesador *p* en un grupo *g* detecta una falla o recuperación, envía una invitación a todos los demás procesadores, a unirse a un nuevo grupo *g'={g.n+1,p}*. Un procesador *q* que recibe tal invitación responde "acepto", si no conoce de otro intento de crear otro grupo *g"* , tal que *g" > g'*. En caso de que *q* haya recibido la invitación para un grupo *g" > g'*, *q* le retransmite a *p* tal invitación. De esta manera se asegura que si más de un procesador intenta crear un nuevo grupo, todos los procesadores converjan hacia un único grupo, que será el de mayor identificador.
En el mensaje de aceptación se envía como parámetro el grupo predecesor que fue abandonado al momento de recibir la invitación.
El procesador que envió la invitación para un nuevo grupo espera 2δ unidades de tiempo, y si en ese intervalo no tomó conocimiento acerca del intento de crear otro grupo con un identificador mayor, entonces concreta la creación del grupo. A tal fin computa la membresía del nuevo grupo *g'*, en base a los procesadores que han aceptado la invitación, y envía un mensaje de "unirse" con parámetros *g'*, *members(g')* y *pred(g')* a todos los demás procesadores de *g'*. El creador de *g'* reune la información sobre los predecesores, contenida en los mensajes de aceptación, y define *pred(g')*; por su parte, los demás procesadores cuando reciben el mensaje "unirse" adoptan *pred(g')* como su predecesor, siempre que *g'* sea el grupo con mayor identificador del que tienen conocimiento.
Un procesador *q* que aceptó la invitación de unirse a *g'* inicia un timer de 3δ unidades de tiempo; si este timer vence y *q* no ha recibido el mensaje para unirse a g', *q* asume que el creador ha fallado, y entonces inicia la creación de otro nuevo grupo *g"={g'.n+1,q}*

4.3 Pros y contras.

Se pueden enumerar los siguientes puntos a favor de este protocolo de membresía:

- Con respecto a otros protocolos, se crean menos grupos transitorios, y el sistema se estabiliza más rapidamente. En [2] se demuestra que:

$$J = D = 9\delta + \text{máx}(\pi + (n+3)\,\delta, \mu)$$

Donde:

J : tiempo que media entre el reinicio (o reconexión) de un procesador y el instante en que todos los procesadores del nuevo grupo concretan la unión al mismo.

D: tiempo que media entre la caída (o desconexión) de un procesador y el instante en que todos los procesadores del nuevo grupo concretan la unión al mismo.

- Existe un tiempo máximo D', que es el retardo para la detección de inconsistencias. Si por ejemplo hay dos procesadores *p* y *q* unidos al grupo *g*, y *q* abandona el grupo, D' es el tiempo máximo que *p* puede suponer que está unido al mismo grupo que *q*, cuando en realidad *q* no está unido al mismo ($D' = \pi + n\delta$).

- El tiempo máximo para la elección del líder, a partir de que el sistema se estabiliza, es: $E = J + D'$.

- Debido a (Sp) los miembros de un nuevo grupo pueden *detectar* inconsistencias con respecto a las historias de los demás miembros del grupo.

Por otra parte, a pesar de las ventajas mencionadas, se puede ver que este protocolo no es lo suficientemente estricto en algunas cuestiones. En el siguiente ejemplo se trata de implementar una política de manejo de la disponibilidad según el criterio de que: "cuando un procesador falla, el *siguiente* procesador toma a su cargo las responsabilidades del que falló", donde el *siguiente* procesador es aquel con el identificador menor, que no tenga aún responsabilidades adicionales. Suponiendo que p3 cae, p1 trata de tomar a su cargo las tareas de p3; si luego cae p5, p2 tratará de reemplazar a p5 debido a que p2 es el siguiente luego de p1. Puede notarse que esta política depende de que p1 y p2 concuerden en el orden en que cayeron p3 y p5. Si p2 cree que p5 falló antes que p3, entonces p1 y p2 intentarán ambos tomar a su cargo los servicios que prestaba p3, pensando que el otro tomará los servicios de p5. En un sistema asincrónico puede ser imposible determinar con exactitud el orden en que suceden dos fallas muy cercanas en el tiempo; sin embargo, esto no es necesario, sino que solamente es necesario que todos los procesadores coincidan en un cierto orden. Desafortunadamente, el próximo ejemplo muestra que con los requerimientos planteados, este protocolo no permite resolver este problema. Supóngase la sucesión de eventos que se detalla a continuación:

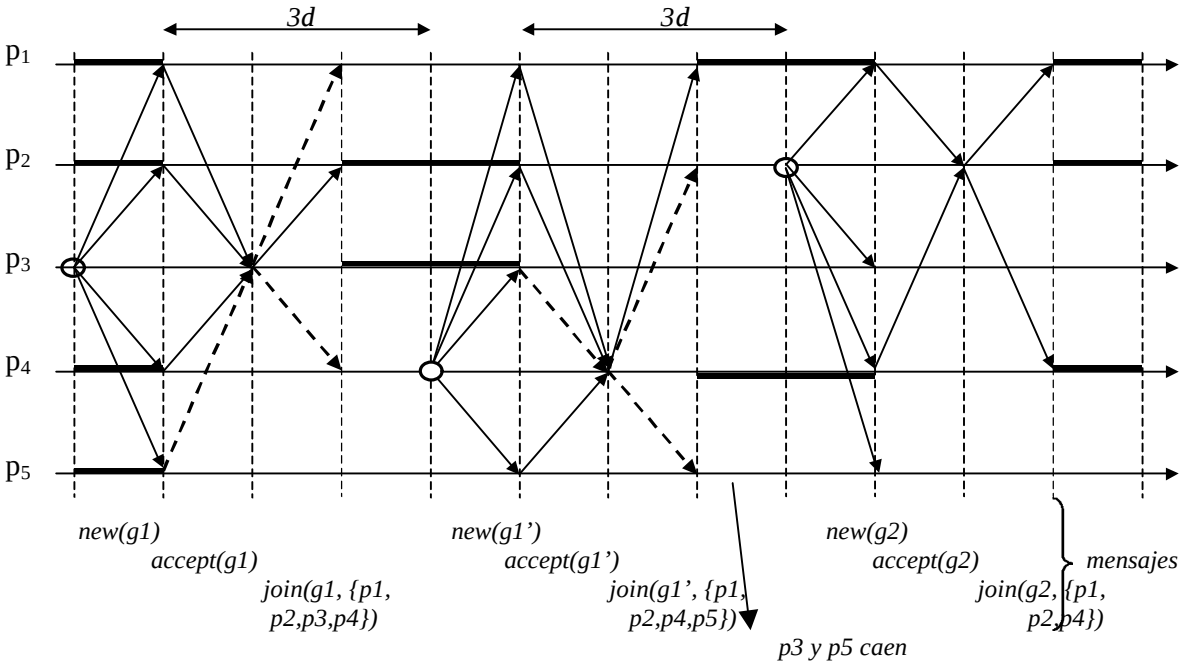
- Inicialmente todos -los cinco- procesadores está unidos al grupo g0={p1,p2,p3,p4,p5}
- Debido a un mensaje "estoy-vivo" perdido, p3 propone la creación de g1.
- Todos los procesadores aceptan, pero la respuesta de p5 se pierde. Entonces p3 ordena unirse al grupo g1={p1,p2,p3,p4}, pero los mensajes de "unirse" no llegan a p1 y p4, por lo que sólo p2 y p3 efectivamente se unen a g1.
- Luego de 3δ unidades de tiempo p4 detecta que no le ha llegado el mensaje de "unirse", y entonces propone crear g1'. Todos los procesadores reciben el mensaje y aceptan, pero se pierde la respuesta de p3. Entonces, p4 ordena unirse al grupo g1'={p1,p2,p4,p5}, solamente p1 recibe el mensaje y los únicos que efectivamente se unen a g1' son p1 y p4.
- Caen p3 y p5.
- Luego de 3δ unidades de tiempo p2 detecta que no recibió el mensaje para unirse a g1' y entonces propone crear g2, todos aceptan y ninguna respuesta se pierde; entonces p2 ordena unirse a g2={p1,p2,p3,p4,p5}, y finalmente el sistema se estabiliza.

En el siguiente gráfico puede verse que la sucesión de grupos a los que estuvieron unidos p1 y p2 son:

p1 : (g0,{p1,p2,p3,p4,p5), (g1',{p1,p2,p4,p5), (g2,{p1,p2,p4))

p2 : (g0,{p1,p2,p3,p4,p5), (g1,{p1,p2,p3,p4), (g2,{p1,p2,p4))

Es decir: p1 supone que p3 falló primero, mientras que p2 supone que p5 falló primero.



Nota: - - - - - → significa mensaje perdido.
O significa que el procesador inicia la creación de un nuevo grupo.
——— significa que el procesador está unido a un grupo. **Fig 4.1**

Los requerimientos planteados sólo aseguran que para los grupos a los que estuvieron unidos p1 y p2, el orden en que se unieron sea el mismo (g0, g2). Sin embargo, no se asegura que p1 y p2 estuvieran unidos a mismo conjunto de grupos, a pesar de que tanto p1 como p2 fueron miembros de todos los grupos que se crearon en el ejemplo. Es decir, a pesar de que el protocolo de tres ruedas con detección de partición permite detectar las inconsistencias (En el ejemplo, cuando p1 y p2 se unen a g2, g1=pred(g2,p2) ≠ g1'=pred(g2,p1), no permite prevenirlas.

5. Protocolo de membresía de tres ruedas para grupos mayoritarios.

El ejemplo recién desarrollado muestra como la ejecución del protocolo de tres ruedas con detección de partición puede llevar a que dos procesadores, p_1 y p_2, vean diferentes historiales de grupos, lo que conduce a conclusiones contradictorias en cuanto al orden en que fallaron p_3 y p_5. Ambas historias contienen solamente grupos mayoritarios y terminan en el mismo grupo final, g_2, pero pasando por diferentes grupos intermedios g_1 y g_1'. Si bien se puede detectar esta inconsistencia, el protocolo no permite resolverla. Se podría reforzar el requerimiento (Sp) a fin de que el protocolo pueda resolver las inconsistencias. En tal sentido, y siguiendo el ejemplo planteado, el protocolo deberá decidir entre g_1 y g_1' cual de los dos grupos es el predecesor "oficial" de g2. La manera natural para tomar esta decisión es en base al orden "<" de los identificadores de grupo; es decir, se elige como predecesor al grupo cuyo identificador sea mayor:

$$Pred(g) = máx\{g' \mid g'<g,\ g' \text{ es un grupo mayoritario, y } \exists\ q \in members(g) : q \text{ joined } g'\}$$

Si ningún miembro de *g* estuvo unido a un grupo mayoritario, se define *Pred(g) = 0*.

El protocolo propuesto se basa en lograr el acuerdo sobre la historia lineal de *grupos mayoritarios completos*. El término *completo* significa que todos los miembros del grupo se han unido a él.

5.1 Requerimientos.

El siguiente requerimiento (Sp') representa una variante más estricta de (Sp):

(Sp') *Notificación de predecesor para grupos mayoritarios*. Sea p un procesador que se une a un grupo mayoritario g. Al momento de unirse el protocolo debe hacerle conocer a p el grupo mayoritario predecesor de g, Pred(g), además de los predecesores individuales pred(g,q) para todos los $q \in$ members(g).

En el ejemplo anterior, el predecesor de g_2 es g_1', el predecesor de g_1' es g_1 y el de g_1 es g_0, con lo que las inconsistencias quedan resueltas. Sin embargo, si en el ejemplo planteado se hubieran perdido todos los mensajes de "unirse" a g_1, entonces p_3 habría sido el único procesador unido a g_1. De esta manera g_1 no podría ser el predecesor de g_1', ya que p_3 no es miembro de g_1', y por lo tanto ninguno de los miembros de g_1' estuvieron unidos a g_1. En este caso, Pred(g_1')=g_0, con lo que g_0 es el predecesor tanto de g_1 como de g_1'.

Si bien Pred define un único predecesor para un cierto grupo, por otra parte un grupo puede tener más de un sucesor. A fin de definir una única historia lineal entre grupos mayoritarios, en [2] se plantea el siguiente requerimiento:

(So) *Historia Lineal de Grupos Mayoritarios Completos*. Sean g < g' dos grupos mayoritarios completos. Entonces g es un ancestro de g' bajo la relación Pred.

Es decir, g es ancestro de *g'* si *g*=Pred(g') o si *g* es ancestro de Pred(g'). Por definición, el grupo 0 es ancestro de todos los grupos mayoritarios.

Al restringir la relación Pred solamente a los grupos mayoritarios se limita el desacuerdo entre dos ramas de la historia a grupos mayoritarios incompletos, de poca vida. Si fuera esencial que todos los procesadores concuerden sobre el orden en que se producen las fallas y los reinicios, solamente aquellos miembros que sepan que están unidos a grupos completos, podrán actuar sobre la información de cambio de membresía. Como un procesador puede estar unido a un grupo incompleto a lo sumo durante D' unidades de tiempo, ésta es la demora en tomar acciones respecto al cambio de membresía.

El siguiente es un ejemplo de una aplicación de nivel superior que necesita que el servicio de membresía cumpla con (So): sea un servicio de difusión atómica cuya finalidad es mantener copias consistentes del estado de un equipo de servidores utilizando un protocolo de difusión por tren de mensajes de dos vueltas [11]. Este protocolo funciona así: un mensaje llamado "tren grupal" circula entre los miembros del grupo, de manera similar a la lista de asistentes vista anteriormente. Cuando un procesador *p* debe difundir una actualización *u* que debe ser aplicada por todo el grupo, *p* espera la llegada del "tren". Cuando el "tren" arriba, *p* añade *u* al final del mensaje y lo hace circular; cuando el tren llega nuevamente a *p* por segunda vez, *p* retira a *u* del tren. Cada procesador aplica *u* a su copia local luego de saber que los demás la conocen, es decir luego de ver por segunda vez consecutiva el tren. De esta manera una actualización no es aplicada a grupos incompletos, y los grupos incompletos que se den a lo largo de la historia no conducen a estados inconsistentes, porque en ellos no se concreta ninguna actualización.

5.2 Implementación del Protocolo.

El protocolo de membresía de tres ruedas con detección de partición se puede modificar para cumplir con el requerimiento (Sp'). La modificación es bastante simple y consiste en que cuando un procesador responde aceptando el mensaje de "nuevo-grupo", además de informar el identificador del grupo al que actualmente está unido, también informe el identificador del último grupo mayoritario del que formó parte. El creador del grupo *g* procesa la información de los predecesores a fin de detectar particiones lógicas, y además establece como Pred(g) al grupo mayoritario cuyo identificador sea el mayor de entre los informados por los procesadores. Finalmente, el creador incluye Pred(g) dentro del mensaje de

"unirse" que le envía a los restantes procesadores del nuevo grupo, y este será el predecesor oficial que adoptan todos ellos.

5.3 Ventajas y desventajas.

Como se ha visto, el protocolo de membresía de tres ruedas para grupos mayoritarios completos evita que dos procesadores puedan estar en desacuerdo acerca del orden en que se suceden las caídas y/o reinicios de procesadores.
Por otra parte, podría considerarse como una desventaja que los grupos mayoritarios incompletos tengan una vida limitada, y no pueda realizarse ningún trabajo útil en los mismos. Sin embargo, como muchas aplicaciones requieren que todos los miembros del grupo estén activos, esto deja de ser una desventaja.

5. 4 Algunas mejoras introducidas al protocolo.

A fin de facilitar y simplificar el cumplimiento del requerimiento básico de que todos los nodos coincidan en una única historia lineal de grupos mayoritarios completos, hemos realizado algunas modificaciones al protocolo presentado en [2].

Cuando un nodo responde aceptando a la propuesta de creación de un nuevo grupo, informa el identificador y la membresía del último grupo mayoritario completo al que estuvo unido. El creador del nuevo grupo determina cual de todos los grupos mayoritarios completos es el más "moderno", y por lo tanto el predecesor oficial del nuevo grupo. En el mensaje de "unirse" (join), el creador envía los siguientes parámetros: identificador del nuevo grupo, miembros del nuevo grupo, identificador del grupo mayoritario completo predecesor, y los miembros de este último grupo. En base a esta información, cada nodo, antes de unirse al nuevo grupo, realiza un análisis que tiene las siguientes posibilidades:

- Si el grupo mayoritario completo predecesor "oficial" coincide con el que tiene registrado, sabe que no estuvo particionado del último tramo del historial de grupos mayoritarios completos, y por lo tanto se une al nuevo grupo.
- El grupo mayoritario completo predecesor "oficial" no coincide con el que tiene registrado, pero su identificador de nodo aparece dentro de los miembros del predecesor "oficial" (!!!). Esto se debe a que se separó de aquel grupo sin haberse enterado que el grupo estaba *completo* (ver más adelante), pero alguno de los sobrevivientes de tal grupo lo dio por completo y lo informó al aceptar el nuevo grupo. En este caso, el nodo que detecta esta situación actualiza su historial (da por completo tal grupo e inmediatamente se separa de él), y luego se une al nuevo grupo.
- El grupo mayoritario completo predecesor "oficial" no coincide con el que tiene registrado, ni tampoco su identificador de nodo aparece dentro de los miembros del predecesor "oficial". Esto se debe a que estuvo particionado del último tramo del historial de grupos mayoritarios completos, y por lo tanto debe registrar tal situación. Los servicios de nivel superior (por ejemplo: difusión atómica), a partir de esta situación, deben actualizar su estado para reflejar las operaciones que se han efectuado mientras estuvo separado.

Por otra parte, teniendo en cuenta que los servicios de nivel superior solamente pueden realizar algún trabajo útil mientras se está unido a un grupo mayoritario *completo*, hemos implementado un mecanismo de "unión" en dos etapas:
- En un primer momento, luego de recibe el mensaje ordenando unirse a un grupo mayoritario, se registra el identificador y la membresía del nuevo grupo, pero todavía no se notifica hacia "arriba".
- El segundo paso se da cuando se toma conocimiento de que el grupo mayoritario es completo –según se explica más adelante–, en ese caso se incorpora este grupo a la vista del historial de grupos mayoritarios completos, y se notifica hacia "arriba".

Por el contrario, luego de unirse a un grupo minoritario, no se notifica nada hacia "arriba", y el líder de tal grupo inicia el procedimiento de "sondeo" para intentar reconectarse con un grupo mayoritario.

A fin de reducir el tiempo que necesita un miembro de un grupo mayoritario para saber que tal grupo es completo, hemos desarrollado el mecanismo que se describe a continuación. Por definición, un grupo mayoritario es completo cuando todos sus miembros se han unido a él. Cuando un miembro recibe el segundo mensaje de "Estoy-vivo" luego de haberse unido, toma conocimiento de que la primera vuelta se completó, y que por lo tanto todos los demás miembros se hayan unidos al grupo. La excepción a esta condición se da para el líder, quien al recibir el primer "estoy-vivo" sabe que todos los miembros están presentes, y por lo tanto el grupo es *completo*. Para acelerar este proceso, las dos primeras vueltas del mensaje de "estoy-vivo" no se inician con un período de π unidades de tiempo, sino solamente δ.

En la versión original, todos los nodos, incluido el líder, a fin de detectar la caída o desconexión de algún nodo utilizan un temporizador de π unidades de tiempo, que se reinicia cada vez que se recibe un mensaje de "estoy-vivo". Otra variante introducida consiste en que cuando el líder del grupo inicia una rueda de mensajes de "estoy-vivo", inicie un temporizador de $n*d$ unidades de tiempo, si vencido el mismo no se recibió el mensaje de regreso, se supone que algún nodo no respondió y se inicia la creación de un nuevo grupo. Mientras tanto, los demás miembros utilizan el temporizador original. Esta variante permite reducir el tiempo de detección de una caída, excepto en el caso de caída del líder, en cuyo caso se mantienen las condiciones originales.

6. Evaluación de la performance.

Hemos desarrollado una implementación del protocolo de membresía grupal de tres ruedas para grupos mayoritarios, sobre una red de computadoras personales ejecutando sistema operativo QNX, vinculadas mediante un segmento Ethernet de 10 Mbps. El código ha sido desarrollado en lenguaje C, utilizando sockets para implementar la comunicación mediante datagramas de tipo UDP. Nuestra implementación consta de tres tareas concurrentes, una dedicada a la recepción de datagramas, otra al envío de datagramas, y la principal encargada del procesamiento de todos los mensajes y eventos. En todos los ensayos la cantidad de procesadores ha sido n=3. El valor de δ que se ha utilizado es de 100 mseg; previamente se determinó que el mínimo δ para que el sistema permanezca estable (en ausencia de fallas de comunicación ni de procesadores) es de 80 mseg. Analizando la composición de este tiempo se ve que la mayor parte se debe al procesamiento en cada nodo, mientras que es mínimo el retardo debido a la red; por lo tanto, es de esperar que con mayor potencia de procesamiento este tiempo se pueda reducir. El valor de π se fijó en 1 seg.

6.1 Comportamiento ante fallas de comunicación.

En esta serie de ensayos no se consideraron caídas de procesadores, sino sólo pérdidas de mensajes. Se simularon diferentes tasas de errores de comunicación: 1 mensaje perdido cada 10, 100, 1000 y 10000 mensajes. Esto se hizo descartando al azar algún mensaje recibido, utilizando rangos de números aleatorios de 10, 100, 1000 y 10000 números respectivamente.

Considerando que los ensayos se realizaron sobre una red de tipo 802.3, con tramas mínimas de 512 bits, la equivalencia entre la tasa de error de bits (BER) y la tasa de pérdida de mensajes es la representada en la siguiente tabla.

	1 msg/10	1 msg/100	1 msg/1.000	1 msg/10.000
BER	2×10^{-4}	2×10^{-5}	2×10^{-6}	2×10^{-7}

Cabe tener en cuenta que las redes actuales tienen tasas de error muy bajas, menores que 10^{-6}, típicamente del orden de 10^{-8}, por lo que es de esperar que en una situación real los valores a esperar serán mejores que los obtenidos para el caso de un mensaje perdido cada 10.000.

En la figura 6.1 se representa la fracción sobre el tiempo total en la que se da la existencia de grupos mayoritarios completos. Se puede ver que para tasas de error bajas o moderadas, el tiempo útil es muy cercano al 100 %. Sin embargo, a menor tasa de error la vida promedio de cada grupo mayoritario

completo es mayor, lo cual es más conveniente para las aplicaciones de nivel superior. Cabe tener en cuenta que luego de la unión a un nuevo grupo, las aplicaciones de nivel superior (por ejemplo: un protocolo de difusión atómica) deben ejecutar algún procedimiento para reconciliar estados entre las distintas réplicas, lo cual consume un cierto tiempo. Por lo tanto, conviene que los grupos tengan una vida lo más larga posible.

En la fig. 6.2 se muestra la duración promedio de un grupo mayoritario completo, en función de la tasa de mensajes perdidos. Esto da una cierta idea acerca de la estabilidad del sistema ante la pérdida de mensajes. Cabe esperar que, en condiciones de normal funcionamiento, y con las tasas de error típicas, los grupos mayoritarios completos tengan una vida promedio no inferior a una hora.

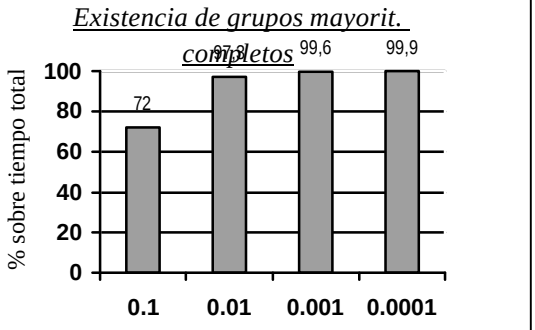


Fig. 6.1

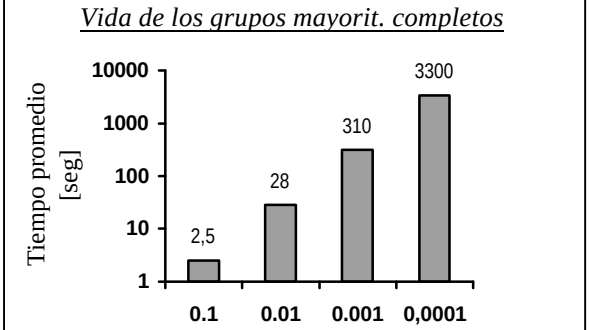


Fig. 6.2

Otro factor que se evaluó es el tiempo (llamémoslo *Dj*) que media entre el instante en que un miembro abandona un grupo mayoritario completo, y el instante en que este mismo nodo determina que se ha *completado* el siguiente grupo mayoritario. Los valores observados a lo largo de diferentes ensayos dan como resultado un tiempo promedio de alrededor de 630 mseg, el mínimo fue de unos 270 mseg, y el máximo es muy dependiente de la tasa de error. Con tasas de error típicas (1 mensaje perdido cada 1000, o mejor), el máximo observado estuvo en el orden de 1,2 seg.

Por otra parte, la cantidad de mensajes por unidad de tiempo que este protocolo le impone a la red es inversamente proporcional al valor de π: M [msg/seg] = *n* [nodos] / π [seg]. Considerando una red de tipo 802.3 de 10 Mbps, cuya trama mínima es de 512 bits, la carga que se le impone a la red es muy baja; esto se detalla en la siguiente tabla para *n*=3 procesadores.

	π = 0.25 [seg]	π = 0.5 [seg]	π = 1 [seg]	π = 5 [seg]
Msg / seg	12	6	3	0.6
Bps	6144	3072	1536	307
Carga de la red [%]	0.06 %	0.03 %	0.015 %	0.003 %

6.2 Reconfiguración ante caídas y reinicios.

Si un nodo cae o se desconecta, el tiempo máximo para detectar tal situación es de π unidades de tiempo. A partir del instante en que un miembro detecta la caída o salida de otro, se inicia la creación de un nuevo grupo, con los valores de D*j* indicados en el punto anterior. Por ejemplo, para π = 1 seg, y Dj_{prom} = 630 mseg, el tiempo promedio entre que la caída o desconexión de un miembro, y la conformación del grupo mayoritario completo siguiente será de alrededor de 1,63 seg.

Por otra parte, el tiempo que media entre el reinicio (o reconexión) de un nodo, y su unión a un nuevo grupo está dado por el valor de μ (período para el envío de mensajes de *"sondeo"*). El valor de μ no

debería ser inferior a 2δ, a fin de dejar el tiempo suficiente para que algún otro nodo reciba este mensaje y responda con un mensaje de creación de un "nuevo grupo".

7. Conclusiones.

En base a los ensayos realizados se pudo observar el buen comportamiento general de este protocolo de membresía. Una variante posible, a fin de aumentar la robustez del sistema, podría ser la siguiente: si bien en las redes actuales es poco probable la inestablidad debida a la pérdida de mensajes, debido a diversos problemas (por ej.: se daña un hub) la red podría caer. Una solución puede ser trabajar con dos segmentos de red duplicados (fig. 7.1), transmitiendo simultáneamente por ambos segmentos, y utilizar algún esquema de numeración de mensajes que permita identificar y descartar los mensajes repetidos que se recibirán cuando ambas redes se encuentren operativas.

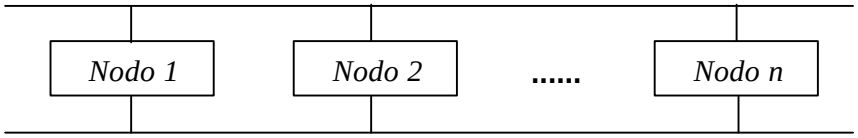


Fig. 7.1

Referencias

[1] Flaviu Cristian & Christof Fetzer, "The Timed Asynchronous System Model", UCSD Technical Report CSE97-519, 1997.

[2] Flaviu Cristian y Frank Schmuck, "Agreeing on Processor Group Membership in Timed Asyncronous Distributed Systems", UCSD Technical Report CSE95-428, 1995.

[3] Rachid Guerraoui y André Schiper, "Fault Tolerance by Replication in Distributed Systems", Proc. Reliable Software Technologics, Ada-Europe'96, Springer Verlag, LNCS 1088, 1996.

[4] Rachid Guerraoui y André Schiper, "Software Based Replication for Fault Tolerance", IEEE Computer, April 1997, pp. 68-74.

[5] Flaviu Cristian, "Understanding Fault-Tolerant Distributed Systems", Comm.of ACM, 34(2); pp 56-78, Feb 1991.

[6] Flaviu Cristian, "Understanding Fault-Tolerant Distributed Systems", Technical Report, CSE - UCSD, May 1993.

[7] Kenneth P. Birman, "The Process Group Approach to Reliable Distributed Computing", Communication of the ACM, Vol. 36, Nº 12, pp. 37-53, Dec. 1993

[8] Flaviu Cristian, Richard de Beijer y Shivakant Mishra, "A Performance Comparison of Asynchronous Atomic Broadcast Protocols", Distributed Systems Engineering Journal, Vol.1, Nº 4, 1994, p.177-201

[9] Flaviu Cristian, Shivakant Mishra y Guillermo Alvarez, "High-Performance Asynchronous Atomic Broadcast", UCSD Technical Report CSE95-450, 1995.

[10] Flaviu Cristian, "Synchronous and Asynchronous Group Communication", Communications of the ACM, April 1996, Vol. 39, Nº 4, pp. 88-97.

[11] Flaviu Cristian, "Group, Majority and Strict Agreement in Timed Asynchronous Distributed Systems", Proc. FTCS, Japan, 1996.