Un sistema distribuido tolerante a fallas basado en protocolos de membresía y difusión atómica.

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

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

Resumen

Los sistemas distribuidos tolerantes a fallas típicamente utilizan alguna estrategia de replicación de servicios en diferentes nodos, a fin de que poder sobrevivir a la caída de alguno de ellos. A fin de simplificar la programación de tales sistemas se considera que los procesadores forman un grupo, y se utiliza entonces un servicio de membresía grupal y un servicio de difusión atómica. El servicio de membresía grupal brinda acuerdo sobre los grupos de servidores que han prestado un determinado servicio a lo largo del tiempo, mientras que el servicio de difusión atómica brinda acuerdo sobre el historial de actualizaciones de estado aplicadas en tales grupos. El presente trabajo describe la implementación de un sistema distribuido tolerante a fallas, a partir de un equipo de computadoras conectadas en red. A fin de asegurar la consistencia entre réplicas, solamente se permite aplicar actualizaciones dentro de grupos mayoritarios completos. El servicio de membresía grupal se encarga de construir el historial de grupos mayoritarios completos, a fin de detectar si el mismo u otro nodo ha estado separado (particionado) de dicho historial, y tomar las medidas pertinentes.

Palabras clave: sistemas distribuidos, tolerancia a fallas, membresía grupal.

1. Introducción

Una estrategia frecuentemente utilizada para la construcción de sistemas distribuidos tolerantes a fallas consiste en disponer de servicios replicados en varios procesadores [3][4], a fin de que el sistema en su conjunto pueda sobrevivir a la caída o desconexión de alguno de ellos. Si bien esta idea es un buen punto de partida en la búsqueda de la tolerancia a fallas, se puede ver que esta estrategia trae aparejada una cierta cantidad de problemas adicionales a resolver, como ser:

- mantener la consistencia en el estado de las distintas réplicas: una actualización debe ser aplicada en todas las réplicas activas o en ninguna. Además, si alguna réplica se separa del

resto (ya sea por caída o desconexión) debe detectar tal situacion a fin de que, al reconectarse, reconcilie su estado con el de las demás.

- si un nodo cae o se desconecta del resto, los demás deben ponerse de acuerdo acerca de quien toma a su cargo las funciones que aquel deja de cumplir (por ejemplo: en un esquema de primario/backup, cual de las réplicas de backup debe asumir como primario en caso de que éste caiga).

Mientras que el primer problema lo debe resolver un servicio de difusión atómica y el segundo es responsabilidad de un servicio de gestión de la disponibilidad de recursos, ambos requieren la existencia de un servicio de membresía grupal que asegure que todos los procesadores activos coincidan en el historial de grupos de procesadores que han prestado un determinado servicio a lo largo del tiempo. Del mencionado historial se obtiene el orden en que los procesadores se han caído (o desconectado) y el orden en que se han reiniciado (o reconectado). Un servicio de difusión atómica necesita de esta información para determinar si ha estado separado (particionado) y por lo tanto necesita reconciliar su estado con el del resto de los procesadores. Un protocolo para el manejo de la disponibilidad de los recursos podría tener una regla que diga que ante la caída de un procesador, sus funciones sean asumidas por el procesador con el identificador siguiente y que aún no ha asumido responsabilidades adicionales. En este caso, es vital que todos los procesadores vean la misma sucesión de caídas y reinicios, porque si no podría suceder que dos procesadores reemplacen a uno que ha caído, o bien que ninguno lo reemplace.

Un criterio de clasificacion distingue entre servicios de membresía grupal de *partición primaria* y *particionables*. Los primeros permiten cambios en la membresía en a lo sumo una única partición física -denominada partición primaria- e intentan asegurar la existencia de un único grupo en cualquier instante de tiempo. Los segundos admiten la existencia simultánea de múltiples grupos, y permiten que un grupo se divida en grupos más pequeños -por ejemplo ante una falla de comunicación- y también que dos o más grupos se unan para formar un nuevo grupo -por ejemplo cuando la comunicación se restablece-.

Según el análisis efectuado por Cristian[7], los protocolos de difusión atómica pueden clasificarse en protocolos de acuerdo *grupal*, acuerdo *mayoritario* y acuerdo *estricto*. Los primeros permiten que en un mismo instante de tiempo se efectúen actualizaciones en dos o más grupos que se encuentran particionados; como consecuencia de esto puede ser necesario un trabajo manual para reconciliar los estados de los distintos grupos al momento de combinarse en uno solo. En los otros dos casos solamente se permiten las actualizaciones en grupos mayoritarios completos[2], por lo que el trabajo de reconciliación es mínimo (acuerdo mayoritario) o nulo (acuerdo estricto).

Los protocolos de difusión atómica de acuerdo grupal requieren la existencia de un protocolo de membresía grupal de tipo particionable -por ej.: tres rondas con detección de partición[2]-, mientras que los protocolos de difusión atómica de acuerdo mayoritario y acuerdo estricto requieren un protocolo de membresía grupal del tipo de partición primaria -por ej: tres rondas para grupos mayoritarios completos o cinco rondas-.

Con el objetivo de avanzar en la implementación de un sistema distribuido con capacidad de tolerancia a fallas, se han analizado diversos protocolos de membresía grupal y difusión atómica, y se ha desarrollado y testeado una variante del protocolo de membresía grupal de tres rondas con detección de partición[8]. En el punto 2 se describen detalles de diseño de la citada implementación y en el punto 3 se discuten los próximos pasos de este proyecto.

En el contexto de los sistemas asincrónicos temporizados [1] se considera que toda la comunicación entre procesos ubicados en diferentes procesadores se realiza mediante mensajes (datagramas), los cuales pueden perderse o demorarse indefinidamente. Por lo tanto, la comunicación mediante mensajes presenta una semántica de falla de *omisión/performance*, si bien la gran mayoría de los mensajes llegará a destino dentro de un cierto tiempo máximo conocido *d*. Por otra parte, como los procesos y procesadores utilizan mecanismos de auto-verificación, es poco probable que entreguen resultados erróneos, por lo que presentan una semántica de falla del tipo de *caída/performance*.

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 período relativamente cortos de inestabilidad. Por lo tanto, y debido a que este tipo de sistemas se puede construir integrando hardware y software estándar, es posible implementar sistemas distribuidos tolerantes a las fallas a un costo relativamente bajo.

A fin de facilitar la lectura del resto del trabajo, a continuación se define una serie de términos que son utilizados más adelante:

– Equipo (de procesadores): es el conjunto total de procesadores.
– Grupo (de procesadores): es el subconjunto de procesadores del equipo que están levantados y conectados entre si, y que han acordado formar parte de un grupo determinado.
– Grupo mayoritario: es aquel grupo compuesto por más de la mitad del total de procesadores ((*N*+1)/2 para *N* impar).
– Grupo mayoritario completo: es aquel grupo mayoritario al que se se han unido todos sus miembros (en contraposición, un grupo mayoritario incompleto es aquel en el cual al menos uno de sus miembros, luego de haber aceptado formar parte del grupo, no ha concretado su unión al mismo -cabe acotar que estos grupos son de muy breve duración-).

2. Descripción del servicio de membresía grupal.

El servicio de membresía grupal ha sido diseñado como una capa dentro de una pila de protocolos, según se ve en la figura 1. Ha sido desarrollado como una variante del protocolo de membresía grupal de tres rondas para grupos mayoritarios desarrollado por Cristian y Schmuck[2].

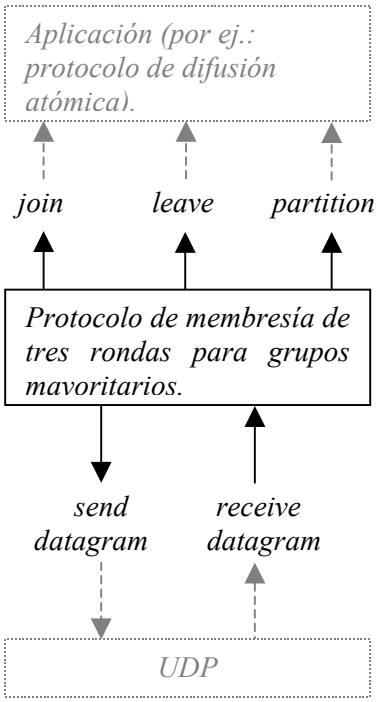

Fig. 1

El servicio de membresía hace uso de las facilidades de envío y recepción de datagramas provista por UDP (User Datagram Protocol), y ha sido implementado mediante sockets. El servicio de membresía se encarga de notificar hacia arriba los diferentes eventos que pueden ocurrir:

- unión a un nuevo grupo mayoritario completo (*join*),
- separación del último grupo mayoritario completo (*leave*),
- detección de partición con respecto al historial de grupos mayoritarios completos (*partition*).

2.1 Operación del protocolo.

Cuando se inicia la creación de un nuevo grupo, el creador envía mensajes *NEW_GROUP* a todos los demás miembros del equipo, y espera durante un tiempo máximo 2δ la llegada de mensajes *ACCEPT* en respuesta al anterior. El mensaje *NEW_GROUP* lleva como parametro el identificador del nuevo grupo cuya creación se propone. El identificador de grupo es un número entero dividido en dos partes: la más significativa es un número que crece monotónicamente a medida que nuevos grupos van siendo creados, y la menos significativa corresponde al identificador del procesador que propone la creación del grupo. En la figura 2 se muestra la definición -en lenguaje C- del tipo de variable utilizado para los identificadores de grupo.

```
/* Tipo para los identificadores de grupo */
typedef union
{
      struct
      {
            unsigned p:4;     /* nro. de procesador (0 a 15) */
            unsigned n:28;    /* nro. id del grupo */
      }x;
      unsigned gid;
}GID_t;
```

Fig. 2

Con este formato del identificador de grupo, es imposible que dos procesadores intenten crear un grupo con el mismo identificador. Asimismo, si dos o más procesadores intentan crear un nuevo grupo casi simultáneamente, todos convergerán al grupo de mayor identificador. En la figura 3 se muestran algunas porciones del código fuente en C, correspondientes a las acciones que la definición -en lenguaje C- del tipo de variable utilizado para los identificadores de grupo.

```
switch(rmsg.cod)
{
case NEW_GROUP:
      /* Se ha recibido un mensaje que propone crear un nuevo grupo */
      if( pv→MaxKnownGid.gid < rmsg.new_group.newid.gid )
      {
            /* El id del nuevo grupo propuesto es mayor que el máximo
               conocido, y por lo tanto es aceptado */
            .......
            /* prepara el mensaje de aceptación */
            smsg.cod = ACCEPT;
            .......
            /* Obtiene el identificador del creador */
            p[0] = rmsg.new_group.newid.x.p;
            .......
            /* Envía el mensaje de aceptación */
            sendUDP( p, &smsg, sizeof(smsg) );
      }else{
            if( pv→MaxKnownGid.gid > rmsg.new_group.newid.gid )
            {
                  /* Le informa a quien le envio este mensaje que ya
                     existe un grupo de mayor ID que el que está
                     intentanto crear */
                  smsg.cod = NEW_GROUP;
                  smsg.new_group.newid.gid = pv→MaxKnownGid.gid;
                  p[0] = rmsg.new_group.q;
                  .........
                  sendUDP(p, &smsg, sizeof(smsg));
            }
      }
      break;
```

Fig. 3

Los parámetros del mensaje *ACCEPT* son:

- identificador del nodo que responde.
- identificador del nuevo grupo al que se está aceptando pertenecer.
- identificador del último grupo mayoritario *completo* al cual estuvo unido (en adelante llamado "predecesor").
- lista de miembros del predecesor.

El creador selecciona el predecesor más nuevo (el de mayor identificador), el que deberá ser considerado por todos los miembros del nuevo grupo como el predecesor "oficial". A continuación el creador envía el mensaje *JOIN*, mediante el cual le ordena a los todos los nodos que han aceptado la propuesta, que concreten la unión al nuevo grupo. Este mensaje contiene los siguientes parámetros:

- identificador del nuevo grupo
- lista de miembros del nuevo grupo
- identificador del grupo mayoritario completo considerado predecesor "oficial"
- lista de miembros del predecesor "oficial"

Luego de recibir el mensaje *JOIN*, cada uno de los miembros del nuevo grupo efectúa el siguiente análisis:

- Si el nuevo grupo es mayoritario:
  (a) Si su predecesor coincide con el predecesor "oficial", asume que está en continuidad con el último tramo del historial de grupos mayoritarios completos.
  (b) Si su predecesor no coincide con el "oficial", hay dos posibilidades:
    (b.1) Su identificador figura en de la lista de miembros del predecesor oficial. Esto se debe a que abandonó dicho grupo un instante antes de haberse enterado que el mismo estaba completo, pero algún otro miembro sí alcanzó a registrar que tal grupo estaba completo. En este caso, se notifica "hacia arriba" la unión e inmediata separación a dicho grupo, a fin de que el nivel superior mantenga un historial consistente, y pueda tomar sus propias decisiones correctamente.
    (b.2) Su identificador no figura en la lista de miembros del predecesor oficial, lo que significa que estuvo separado de la última parte del historial de grupos mayoritarios completos. En este caso, si se llega a determinar que el grupo está completo, se notifica "hacia arriba" tanto la unión, como así también el hecho de que se está reingresando al historial (*partition* + *join*), y por lo tanto las aplicaciones de nivel superior pueden requerir una conciliación de estados.

- Si el nuevo grupo es minoritario:
  Se concreta la unión al nuevo grupo, sin efectuar ninguna notificación "hacia arriba". El líder de este grupo comienza el proceso de "sondeo" para tratar de retomar contacto con el

resto de los nodos; para ello inicia el envío periódico de mensajes *PROBE*. Cuando el líder de un grupo recibe un mensaje *PROBE* dispara la creación de un nuevo grupo.

2.1.1 Determinación de la completitud de un grupo mayoritario.

Luego de haberse creado un grupo mayoritario, la notificación "hacia arriba" se demora hasta saber que el grupo está completo. Las razones por las que un grupo mayoritario puede quedar incompleto son:

– la pérdida de alguno de los mensajes *JOIN*,
– la caída o desconexión de un nodo inmediatamente después de haber enviado *ACCEPT*.

El mecanismo utilizado para determinar que el grupo se ha completado consiste en hacer circular dos veces seguidas la "*lista de asistencia*". Luego de la primera vuelta el líder de grupo sabe que el grupo está completo -por que todos han retransmitido dicho mensaje-, mientras que el resto de los nodos se enteran que el grupo está completo cuando reciben por segunda el mensaje de la "*lista de asistencia*". Si durante la segunda vuelta se pierde alguno de estos mensajes, algunos nodos considerarán que el grupo está completo mientras que otros no; sin embargo, como la pérdida de dicho mensaje dará lugar a la creación de un nuevo grupo, dicha inconsistencia será resuelta según lo explicado en el párrafo 2.1.

2.2 Detalles del software.

El sistema ha sido estructurado en base a tres tareas que se comunican entre sí por medio de mensajes, siguiendo la filosofía cliente/servidor. El código ha sido escrito en lenguaje C sobre sistema operativo QNX (aunque sería fácilmente migrable a otro sistema operativo). Hay una tarea principal y dos tareas subsidiarias encargadas del manejo de las comunicaciones con otros nodos, tal como se muestra en la figura 2. Las tres tareas que componen el sistema son:

- Tarea principal: es la responsable de la implementación del protocolo de membresía. Procesa todos los mensajes y eventos que ocurren en el sistema, y se encarga de la interfase con la capa superior. Para enviar mensajes a otros nodos, a través de la red, hace uso de la tarea auxiliar *SendUdpServer*. Por otra parte, los mensajes que llegan desde la red son recibidos por la tarea *ReceiveUDP,* quien se los pasa a la tarea principal.
  La tarea principal trabaja básicamente como servidor, estando bloqueada a la espera de mensajes o eventos que la activen.

- Recepción de datagramas: la tarea *ReceiveUDP* es creada durante la inicialización del sistema, y crea un socket para esperar mensajes provenientes de la red. Cada mensaje recibido es reenviado a la tarea principal para su procesamiento.

Desde el punto de vista del modelo cliente/servidor esta tarea puede verse como un servidor, bloqueado a la espera de mensajes provenientes de la red. Sin embargo, como a su vez cada mensaje es reenviado a la tarea principal, *ReceiveUDP* es cliente de la tarea principal.

- Envío de datagramas: la tarea *SendUdpServer* es creada durante la inicialización del sistema, y crea un socket desde el cual enviará datagramas a la red. Cuando la tarea principal debe enviar un datagrama a uno o más nodos, le pasa a la tarea *SendUdpServer* un mensaje consistente en el contenido que debe llevar el datagrama y la lista de identificadores de los nodos a los cuales debe ser transmitido el mensaje. La tarea *SendUdpServer* determina la dirección IP de cada nodo destinatario y le reenvía el mensaje correspondiente.
  Desde la óptica del modelo cliente/servidor, esta tarea trabaja como servidor, a la espera de requerimientos de la tarea principal.

En la figura 4 se muestra la interrelación entre las tres tareas y los mensajes que intercambian entre ellas.

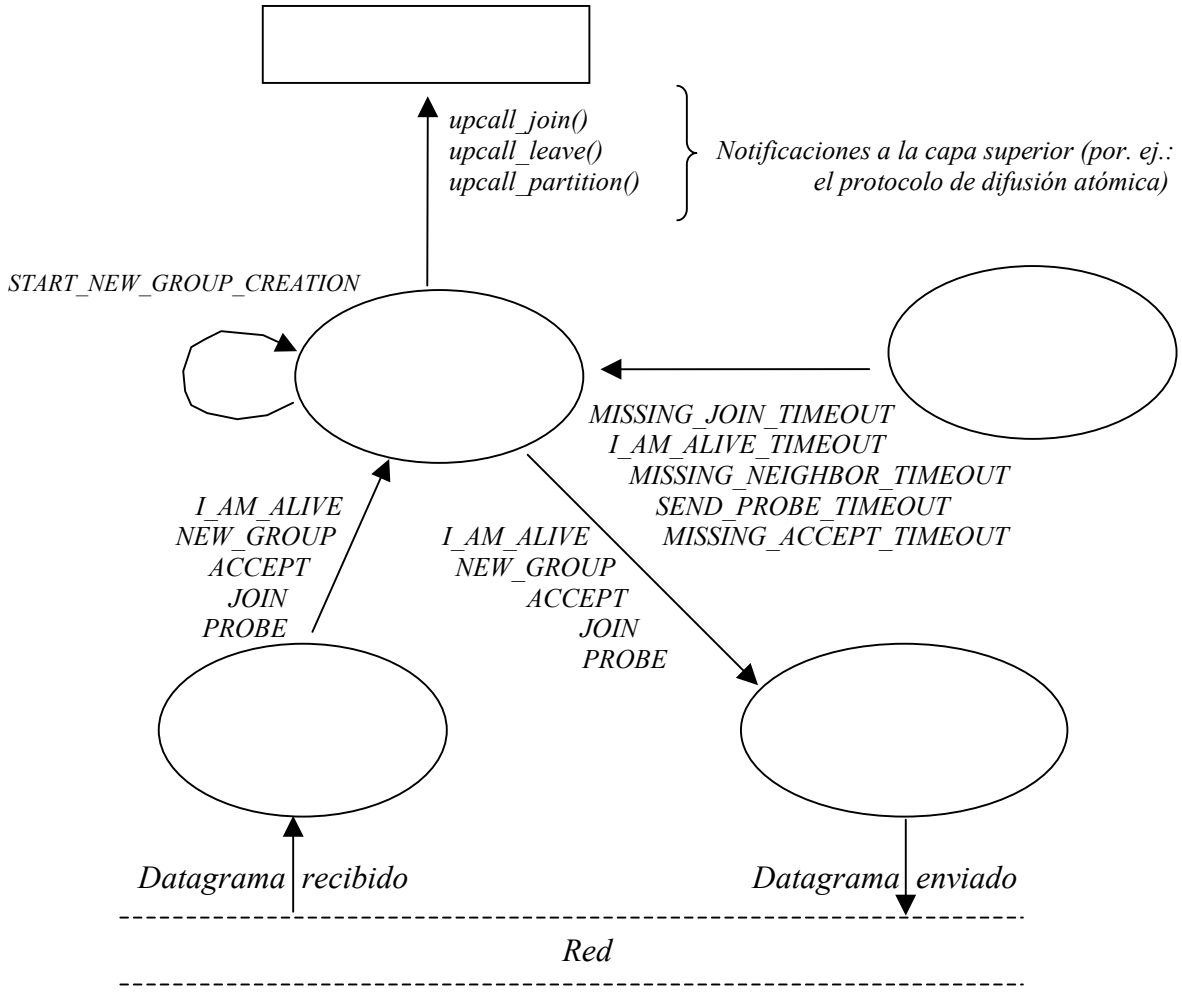

Fig. 4

También se puede ver que hay otros mensajes que recibe la tarea principal, y que son generados al vencer determinados temporizadores. Por otra parte, el evento denominado *START_NEW_GROUP_CREATION*, es generado dentro de la misma tarea principal y enviado a si misma, cuando se detecta la necesidad de disparar la creación de un nuevo grupo. Esto se ha hecho de esta manera a fin de uniformar la recepción de eventos por parte de la tarea principal: todos los eventos se reciben mediante un mensaje.

2.3 Interacción con la capa superior.

Se pueden observar algunas diferencias entre esta implementación y el seudocódigo presentado en[2]: mientras que en [2] se definen tareas independientes para cada una de las funciones del protocolo, en nuesta implementación hay con una tarea principal que procesa todos los eventos y mensajes del protocolo, y un par de tareas auxiliares para el envío y recepción de datagramas. Otra diferencia radica en que nuestra versión del protocolo de membresía recién notifica "hacia arriba" la unión a un nuevo grupo, luego de saber que tal grupo es mayoritario completo. De esta manera, el cliente local (por ejemplo: el servicio de difusión atómica), no se entera de uniones a grupos minoritarios y/o a grupos mayoritarios incompletos –de poca vida–, y que no permiten realizar trabajo útil. El cliente local puede recibir alguno de los siguientes mensajes:

- Unión a un nuevo grupo mayoritario completo, con continuidad en el historial de grupos mayoritarios completos (*upcall_join()*).

- Unión a un nuevo grupo mayoritario completo, pero detectándose que previamente se ha estado particionado de la última parte del historial de grupos mayoritarios completos. En base a esto, el cliente sabe que deberá actualizar su estado (*upcall_partition()* seguido de *upcall_join()*).

- Se ha abandonado el grupo mayoritario completo actual (*upcall_leave()*).

2.4 Propiedades del protocolo de membresía.

Teniendo en cuenta las propiedades que caracterizan a un servicio de membresía[9], en función de que mensajes de notificación envía "hacia arriba" y del momento en que los mismos son entregados -en relación a otros mensajes y al tiempo real-, se puede establecer que el protocolo implementado cumple con las siguientes propiedades:

- Es *vivaz* (pero inexacto) en la detección de caídas. La inexactitud se debe a que debido a la pérdida o demora excesiva de un mensaje se puede sospechar equivocadamente de la falla (o desconexión) de un nodo.

- Es *vivaz* y *exacto* en la detección de recuperaciones. Cuando un nodo se reinicia o reconecta envía mensajes de sondeo, y la recepción de uno de tales mensajes anuncia exactamente la recuperación (o reconexión) de uno o más nodos.
- Particionamiento del tipo de partición primaria.
- Acuerdo limitado a la partición primaria. Los nodos de la partición primaria coinciden en la historia lineal de grupos mayoritarios completos, y en el orden en que se producen las caídas y reinicios de nodos.
- Ordenamiento total de los mensajes de cambio de membresía (dentro de la partición primaria).
- Inicio individual: cada nodo inicia en un grupo de un solo miembro, sin conocer una membresía a priori, e intenta tomar conocimiento de que otros nodos están presentes.
- Unión colectiva: cuando se restablece la conexión entre dos particiones se crea un nuevo grupo que combina ambas membresías, en una sola operación.

Por otra parte, este protocolo le facilita al protocolo de difusión atómica el cumplimiento de las propiedades de:

- *Acuerdo sobre el primer mensaje*: que todos los sitios comiencen la entrega de mensajes provenientes de un sitio nuevo (o recuperado) a partir del mismo mensaje.
- *Acuerdo sobre el último mensaje*: que ante un mensaje de cambio de membresía indicando la salida de un cierto sitio *C*, todos los sobrevivientes coincidan en cual será el último mensaje proveniente de *C* que será entregado. De esta manera, si tal mensaje producía un cambio de estado se asegura que será aplicado por todos o ninguno.
- *Acuerdo sobre los predecesores*: por ejemplo, suponiendo que un sitio *C* falla, el conjunto de predecesores acordados consiste exactamente de aquellos mensajes sobre los que se tiene certeza que han sido recibidos por la capa de comunicación de *C* previo a su falla.
- *Acuerdo sobre los sucesores*: esta propiedad afecta a la estabilidad de los mensajes (un mensaje es estable cuando fue entregado en todos los sitios). Si *r(c)* es un mensaje que notifica la recuperación de un sitio *C*, y el mensaje *m* es un sucesor acordado de *r(c)*, entonces todos los nodos saben que *m* deberá ser reconocido por *C* para poder ser considerado como *estable*.
- *Sincronía virtual*: esta propiedad requiere que todos los sitios activos estén de acuerdo en la división del flujo de mensajes, de tal modo que cada mensaje o bien sea un predecesor acordado o bien sea un sucesor acordado. Es decir, la sincronía virtual crea un *corte acordado* en el flujo de mensajes.
- *Sincronía virtual extendida*: esta propiedad garantiza que los mensajes entregados bajo la antigua membresía sean entregados siempre antes que el mensaje de cambio de membresía. Por ejemplo: si un nodo *A* envía un mensaje a_i cuando la membresía es $\{A,B,C\}$, y otro nodo *D* se agrega al grupo antes de que el mensaje a_i sea recibido, la sincronía virtual extendida asegura que a_i será recibido por B y C antes que el mensaje de cambio de membresía, mientras que no será recibido por *D*.

Para lograr el cumplimiento de estas propiedades, el servicio de membresía notifica al servicio de difusión atómica el identificador y la membresía del grupo actual. Los mensajes de aplicación llevan la marca del grupo al que pertenece el nodo de origen al momento del envío. En base a esta información, cuando se forma un nuevo grupo, los servicios de difusión atómica pueden distinguir entre los mensajes enviados dentro del grupo anterior y los mensajes enviados dentro del nuevo grupo, y llegar a un acuerdo para cumplir con las propiedades mencionadas.

3. Conclusiones y estado actual.

Actualmente se está trabajando en el desarrollo de un servicio de difusión atómica, sustentado en el servicio de membresía grupal antes descripto. Un protocolo de difusión atómica tiene la misión básica de asegurar la entrega de un mensaje o bien en todos los nodos o bien en ninguno. Otro requisito adicional -y de gran importancia- para un protocolo de difusión atómica consiste en imponer un mismo orden de entrega de los mensajes en todos los nodos. Hay dos tipos de protocolos que permite cumplir con este requerimiento:

– *Protocolos basados en secuenciador*: uno de los servidores de difusión cumple con la misión adicional de imponer un orden global a todas las actualizaciones provenientes de cualquier servidor.
– *Protocolos basados en pasaje de ficha (o tren)*: hay un mensaje (ficha o tren) circulando entre los miembros del grupo. Cuando un servidor desea difundir una actualización, debe esperar a tener la ficha en su poder. En este caso, la ficha lleva (explícita o implícitamente) la información de secuenciación.

En un trabajo de Cristian y otros [5] se analiza la performance de varios protocolos de estos dos tipos, y se puede ver que ninguno es totalmente superior a los demás en todos los aspectos considerados. Muy brevemente se puede decir que los protocolos basados en secuenciador arrojan los mejores tiempos de entrega y estabilidad -en especial usando una estrategia de reconocimiento positivo (*Positive Acknowledge*)-, mientras que los protocolos de tren requieren menor cantidad de mensajes por difusión, tienen mejor distribución de carga (no hay un nodo recargado por ser secuenciador), y trabajan mejor con altas tasas de actualización. Asimismo, en los protocolos basados en secuenciador, la situación más favorable se da cuando el mismo nodo que genera la actualización es el secuenciador -hay un ahorro de mensajes-.

Por estas razones, en un trabajo posterior [6] Cristian y otros intentaron reunir las mejores propiedades de cada tipo de protocolo en uno solo, y llegaron a dos protocolos de secuenciador móvil. El protocolo *pinwheel* (molinillo) utiliza un secuenciador rotativo y demostró ser mejor cuando la distribución de actualizaciones es uniforme. Por su parte, en el protocolo *on-demand* (por demanda) el rol de secuenciador se va transfiriendo a pedido, a medida que los nodos quieren transmitir actualizaciones, y resultó ser más eficiente cuando los nodos generan ráfagas de actualizaciones (un comportamiento típico en las aplicaciones multimedia). En ambos casos, la idea básica es que el nodo que quiere enviar una actualización debe tener en ese momento el rol de secuenciador.

Nuestro objetivo es desarrollar un protocolo de difusión atómica que tenga una cierta capacidad de adaptarse al tráfico uniforme y por ráfagas. Asimismo se pretende aprovechar la mensajería subyacente del protocolo de membresía grupal, a fin de reducir el tráfico total en la red. En tal sentido, la idea es agregar una carga adicional a los mensajes de la "lista de asistencia", para uso del protocolo de difusión atómica. Toda la mensajería necesaria para secuenciación, reconocimiento de actualizaciones (acknowledges), transferencia del rol de secuenciador, etc. se transportará aprovechando la "lista de asistencia", que ya de por sí circula periódicamente entre los miembros del grupo.

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 Asynchronous 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] F. Cristian, R. de Beijer y S. Mishra, "A Performance Comparison of Asynchronous Atomic Broadcast Protocols", Distributed Systems Engineering Journal, Vol.1, Nº 4, 1994, p.177-201

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

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

[8] Guillermo R. Friedrich y Jorge R. Ardenghi, "El protocolo de membresía grupal de tres ruedas como base de un sistema distribuido tolerante a fallas", Proceedings del VII Congreso Argentino de Ciencias de la Computación, Octubre de 2001, pp. 669-682.

[9] M. Hiltunen y R. Schlichting, "Properties of membership services". Proceedings 2º Int. Symp. on Autonomous Decentralized Systems, Phoenix, AZ, Apr. 1995.