Una plataforma distribuida para sistemas de remates

Federico del Rio Garcia[1],

[1] Sistemas Distribuidos I, Facultad de Ingeniería, Universidad de Buenos Aires, Argentina
f.del.rio.garcia@gmail.com

Abstract. La mayoría de las aplicaciones distribuidas imponen requerimientos, en cuanto a las comunicaciones, que no pueden contemplarse dentro del diseño mismo de una aplicación para usuario final; sino que deben contemplarse en el marco de un sistema distribuido. Este trabajo se focaliza en una plataforma distribuida, dejando de lado la aplicación para el usuario final, y concentrándose en los problemas que plantea el intercambio de información para un sistema de remates.

Una plataforma debe tener en cuenta todos los requerimientos para este tipo de sistemas, que son: formar un grupo para cada remate, proveer comunicaciones confiables con todos los participantes, incorporar y separar participantes de los grupo, mantener el estado del remate consistente (fundamentalmente a través del orden de los mensajes) y permitir implementarlo sobre una WAN. La plataforma más adecuada es un middleware asincrónico persistente orientado a grupos.

Implemento un prototipo del sistema para probar algunos de los puntos más importantes del diseño.

Keywords: sistema distribuido, middleware, comunicación, grupos cerrados, asincrónico, persistente.

1 Introduction

El problema planteado en el trabajo es el diseño de un middleware [1] [2] para un sistema de remates. Middleware es una plataforma de software entre la aplicación y los servicios de comunicación del sistema operativo, para ocultar en mayor o menor grado la heterogeneidad de la colección de sistemas operativos y redes involucradas y para mejorar la transparencia de distribución.

Antes de imponer requisitos o condiciones sobre la infraestructura de red, se debe cumplir con todos los requerimientos de este tipo de sistemas:

- Creación de sesiones de remates (con toda la funcionalidad que esto implica, como incorporar y separar integrantes a las sesiones de remates, etc.).
- Entrega confiable de los mensajes definidos en la aplicación.
- Entrega a todos los participantes pertenecientes a una sesión de remates (difusión).

- Como en cualquier sistema, el estado del mismo debe ser consistente. En este caso particular la consistencia está relacionada con las ofertas que se aceptan durante la sesión de remate, es decir que la última oferta válida debe ser la misma para todos los participantes. Por consiguiente, para que el estado del sistema sea consistente, todos los mensajes deben llegar en el mismo orden.
- Invitar a todos los participantes que están conectados al sistema a unirse a las sesiones de remates.

Del último punto mencionado se desprende la necesidad de un esquema de difusión de mensajes para todos los participantes del sistema. Este esquema de difusión debe alcanzar tanto a los participantes de sesiones de remates existentes (difusión de ofertas), como a los participantes del sistema en general (invitaciones y notificaciones especiales que deba hacer la aplicación).

El rematador es el encargado de administrar las sesiones de remates, creando, iniciando y finalizando las mismas. Durante la creación de la sesión de remate, se invita a todos los participantes, ya que pueden participar de más de un remate a la vez. Los participantes se incorporan a la sesión antes de que se inicie. Entre el inicio y la finalización del remate (sesión) se realizan y aceptan las ofertas, solamente de los participantes que se unieron al remate.

Un remate funciona de manera muy similar a un Grupo Cerrado [1], de un sistema distribuido. Un servicio de grupos provee dos funcionalidades a los participantes: la capacidad de convertirse en miembro de un grupo (membresía), y la de proveer información actualizada sobre el estado del grupo (visión). Se dice que es cerrado cuando envían y reciben mensajes sólo los participantes que poseen una membresía.

Comparando los requerimientos del sistema de remates con el servicio de grupos se ve que: una sesión de remate constituye un grupo determinado, la incorporación y separación de participantes del remate es una membresía y las ofertas son las vistas del grupo. Como sólo pueden ofertar y recibir ofertas los participantes que se incorporaron al remate (o lo que es lo mismo, poseen una membresía), el grupo debe ser cerrado.

El middleware debe ser persistente para garantizar la entrega de mensajes.

El sistema debe funcionar sobre una WAN[1], debido a esto el middleware debe ser asincrónico. Mantener el sincronismo resultaría demasiado lento y costoso por la cantidad de mensajes que se necesitan.

El intercambio entre los participantes que componen el sistema es siempre por medio de mensajes, por lo tanto esta plataforma debe ser orientada a mensajes o MOM (Message Oriented Middleware) [2]. Los sistemas MOM proporcionan un amplio soporte para las comunicaciones asincrónicas persistentes. De esta manera se elige trabajar sobre un paradigma que extiende el modelo de trabajo de la aplicación (figura 1).

Dado que el sistema debe funcionar sobre una WAN, hay dos niveles de brokers. Un Broker o Message Broker, es un intermediario que transmite un mensaje desde el emisor hacia el receptor, transformando el mensaje a los protocolos respectivos de ser necesario. Toda la comunicación entre las distintas LAN[2] se realiza a través de

[1] *Wide Area Network*
[2] *Local Area Network*

Distribuidores Remotos (DR), y la comunicación dentro de cada LAN se realiza a través de Distribuidores Locales (DL) [9] [10].

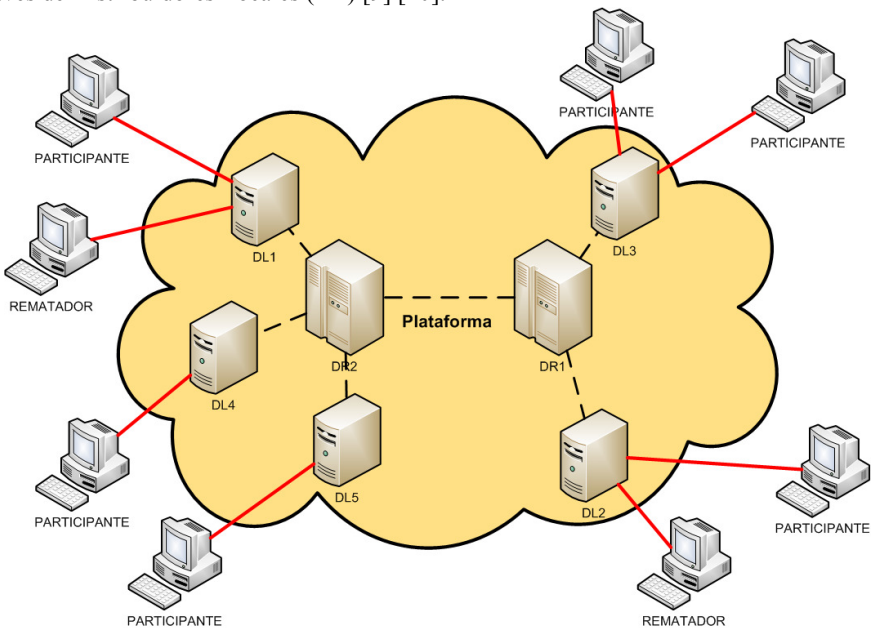


Fig. 1. Vista de la plataforma desde la aplicación. Los brokers pertenecen a la plataforma.

Resumiendo entonces los requerimientos del middleware, debe cumplir con:

- Un esquema de difusión de mensajes.
- Orientado a grupos.
- Los grupos deben ser cerrados.
- Persistente.
- Asincrónico.
- Orientado a mensajes o MOM.

2 Diseño del Sistema

En el sistema se observan dos grandes necesidades, por un lado el esquema de difusión para las comunicaciones y por el otro, brindar soporte para grupos a la aplicación basándonos en el esquema anterior. Debido a esto dividimos el middleware en dos capas: una capa de Multicast (sistema de difusión) [1] y por encima de esta una capa de Grupos. El protocolo de Multicast es el responsable de entregar un mensaje a todos los participantes de un grupo.

Multicast

El servicio Multicast se implementa a nivel de la capa de aplicación OSI [7], este tipo de protocolos de multicast son conocidos como Application Layer Multicast (ALM) [5]. ALM es el nombre que reciben los protocolos diseñados para proveer servicios de multicast en la capa de aplicación del modelo OSI (Open System Interconnection o Interconexión de Sistemas Abiertos).

Como se ve en la figura 2, el esquema de difusión se realiza a través de los distintos brokers. Aprovechando la jerarquía de brokers impuesta por el tipo de sistema, desarrollo un protocolo que toma distintos aspectos de los algoritmos de multicast adaptándose a este problema [5].

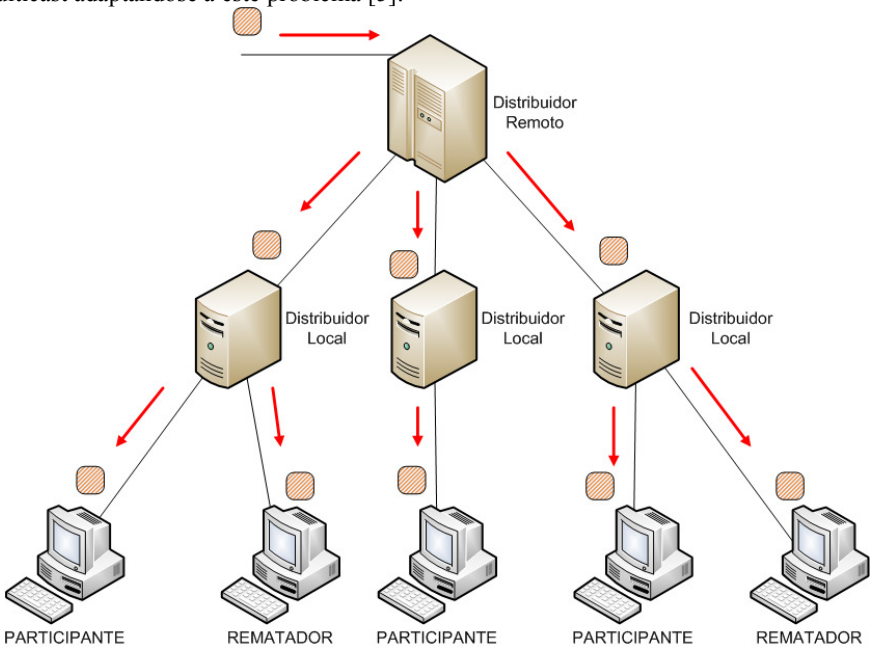


Fig. 1. Esquema de diffusion de un mensaje en la capa.

En esta capa se emplea el concepto de grupo de multicast, esto se refiere a un conjunto de participantes y rematadores o de DRs o de DLs entre los cuales se debe difundir un mensaje. Hay tres tipos de grupos distintos: grupos de Hosts, que están compuestos por un DL y *n* Hosts; grupos de DLs, que están compuestos por un DR y *n* DLs; y grupos de DRs, que están compuestos sólo por DRs (ver figura 3). En los dos primeros el encargado de la difusión es el DL y el DR respectivamente, en el grupo de DRs todos funcionan como pares y cualquiera puede difundir un mensaje. Esto aumenta la complejidad debido a que cuando llega un mensaje a un DR puede ser necesaria la difusión a los otros DRs o no, dependiendo del origen del mismo.

Los grupos son independientes entre sí, de manera que no existe necesidad de llegar a todos los brokers para hacer una difusión a nivel local.

Esta capa ofrece, además, un servicio de entrega punto a punto o Unicast, de forma que si la aplicación requiere enviar mensajes especiales puede usar este mismo sistema.

En esta capa también resuelvo el requerimiento de persistencia. Para esto la entrega de mensajes punto a puntos se hace en forma sincrónica y se almacenan los mismos hasta recibir la confirmación de su llegada [6]. Por ejemplo cuando un DR envía un mensaje a un DL lo almacena, lo envía y espera la confirmación de llegada. Una vez recibida se elimina el mensaje. Además, se provee un esquema de retransmisión para entregar mensajes y detectar conexiones caídas. Una vez restablecida la conexión se recuperan todos los mensajes no entregados.

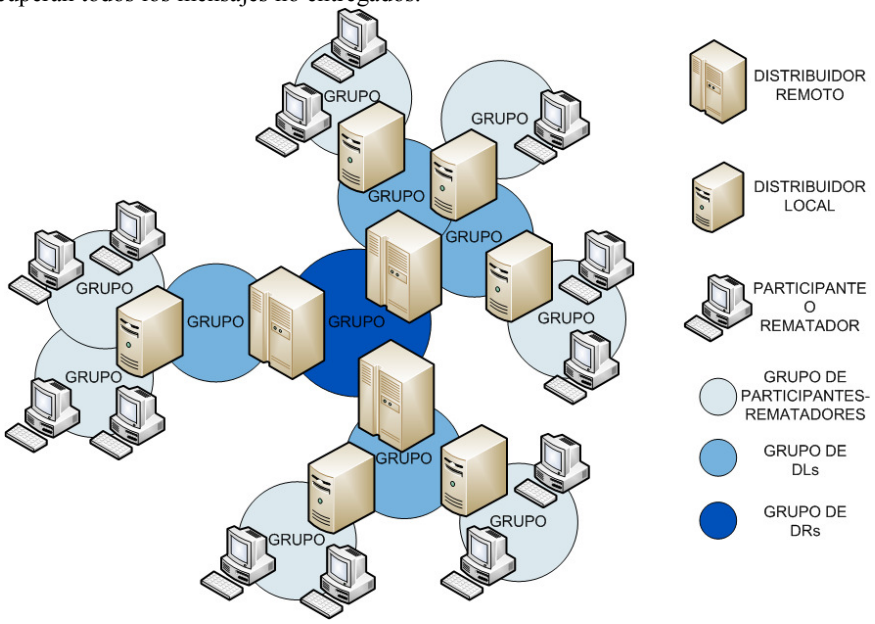


Fig. 3. Distintos tipos de grupos de la capa Multicast.

Esta capa reserva el identificador de grupo *0* para el grupo de DRs compuesto por todos los DR que pertenecen al sistema (a medida que se van incorporando nuevos DR se suman al grupo). Aunque este grupo es para ser utilizado por las capas superiores.

Grupos

Esta capa se ocupa, fundamentalmente, de mantener el estado consistente del sistema. Para ello es necesario que dentros de un grupo los mensajes lleguen en orden total [1]. Hay orden total cuando dos mensajes cualesquiera entregados a cualquier par de participantes son entregados en el mismo orden a ambos; es decir que todos los mensajes llegan en el mismo orden. De la misma forma que lo resuelve van Renesse et al. [8], el problema de orden de los mensajes lo resuelvo en la capa Grupos.

Dado que los mensajes deben circular en una WAN, para lograr orden total escojo el algoritmo de Bakery [3]. Si bien este algoritmo presenta un punto centralizado (y por lo tanto un único punto de falla), es uno de los algoritmos que requiere el menor intercambio de mensajes para lograr este orden total.

Aprovechando el hecho de que los únicos que tienen conocimiento de todos los DRs en el sistema son los mismos DRs, uno de ellos es designado como líder para asignar números de orden a los mensajes y mantener los números de grupos de la capa. Cuando se envía un mensaje al grupo, el DR líder le asigna un número de secuencia y después se hace la difusión a todos los integrantes del grupo.

Para determinar el líder aplico un algoritmo de elección de líder [1]. Este algoritmo necesita que todos los candidatos a tomar la posición de líder puedan intercambiar mensajes. Para este intercambio utilizo el grupo de multicast *0* explicado anteriormente, ya que todos los DRs pueden asumir esta posición. Este algoritmo también requiere orden total de los mensajes. En este caso se utiliza el algoritmo del reloj de Lamport [4]. De este modo el protocolo elige el líder necesario para el algoritmo de Bakery.

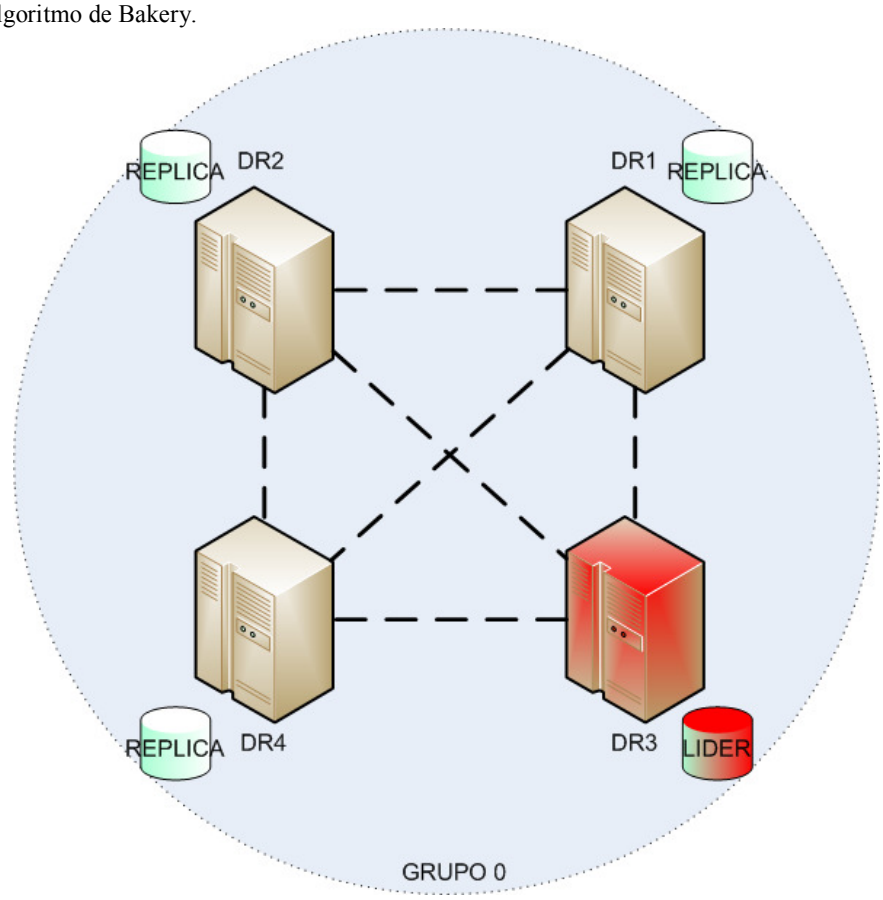
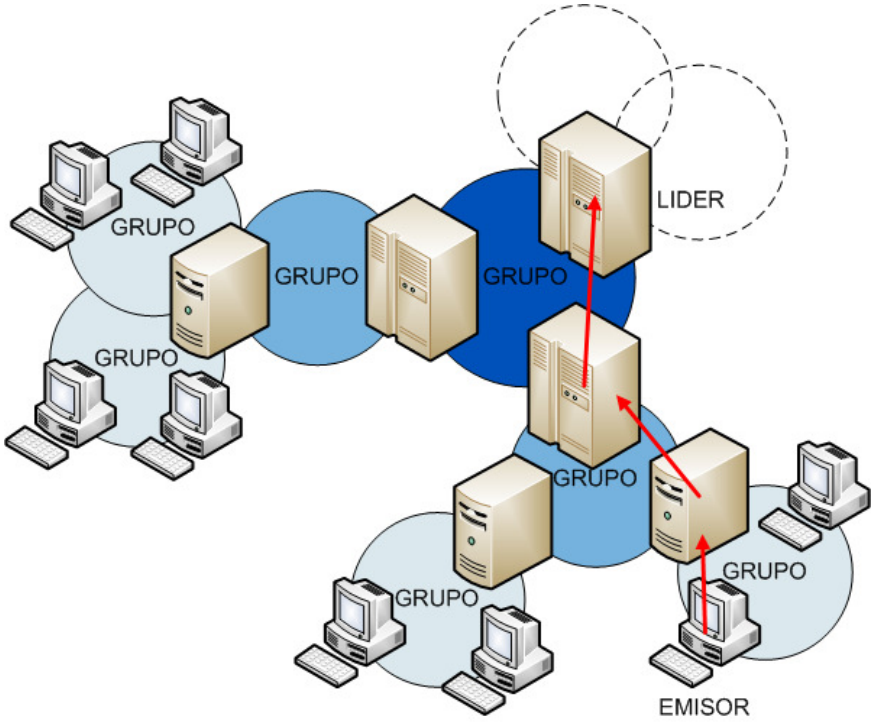


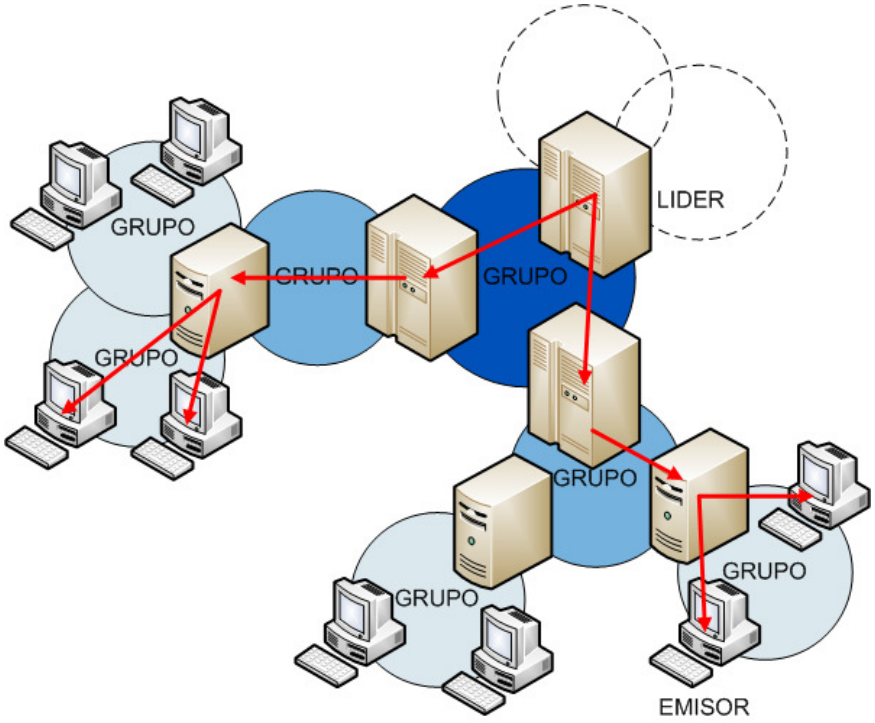
Fig. 4. Grupo de multicast 0, definido por defecto para todos los DRs del sistema. Cada DR mantiene una réplica de la información del líder. Existe también una conexión entre cada uno de los DRs.

Una vez que se determina el líder, este mantiene la información replicada en todos los DR (figura 4). En caso de que se caiga el líder, cualquiera de los otros DRs del sistema puede tomar su lugar disponiendo de toda la información necesaria. Cuando un DR del grupo detecta que se perdió la conexión con el DR líder, inicia nuevamente el algoritmo de elección de líder. Con este protocolo adicional se soluciona el punto de falla mencionado anteriormente.

Cuando un Host publica un mensaje en el grupo, este sigue un proceso como el que se muestra en la figura 5. Se ve que en primera instancia se debe secuenciar el mensaje. El Host envía el mensaje a su DL, este se lo envía a su DR y este último al líder. Una vez que el líder le asigna un número de secuencia lo difunde entre todos los participantes del grupo correspondiente, haciendo el mapeo *grupo* ↔ *grupo de multicast* necesario.



a) Primero el mensaje es numerado por el líder.



b) Después se entrega a todos los participantes del grupo.

Fig. 5. Ciclo de difusión de un mensaje en un grupo.

Resolución de Distribuidores Remotos

Como los Hosts y los DLs se conectan con un solo DL y DR respectivamente, ambos se conectan utilizando la dirección o nombre de DNS [1] (Sistema de Nombre de Dominio o Domain Name System) de los mismos. Los DRs, en cambio, deben conectarse con todos sus pares; para ello necesitan saber cuáles están activos y dónde. Para resolver este problema incorporo a la capa de Multicast un servicio de Resolución de Distribuidores Remotos (SRDR), inspirado en el DNS.

La topología presentada en este trabajo para la WAN, la divide en zonas compuestas cada una por distintas LANs. Entonces a cada zona se le asigna un servidor de SRDR con el que se conecta cada DR de la misma y cada DR conoce la dirección de este. Dado que se conecta por única vez con el SRDR y que el intercambio de mensajes entre estos es muy corto, el protocolo de comunicación es sin conexión sobre UDP.

Al ser una cantidad fija de servidores, estos se conocen entre sí. Cada servidor guarda información acerca de los DRs de su zona que están conectados y también de

los servidores de las otras zonas. Esta última información, la intercambian entre los SRDR mediante un protocolo a tal fin. Cuando se conecta un nuevo DR, su SRDR actualiza la tabla de conexiones y le informa a este de todos los DRs que están conectados. Una vez que el SRDR actualiza su tabla, informa al resto de los SRDR de la nueva conexión, de esta forma no es necesario intercambiar todas las tablas de conexiones entera, de todos los servidores, por cada nuevo DR que se conecta sino que sólo se informan las conexiones nuevas.

3 Conclusión

En este trabajo se presenta un sistema distribuido que soluciona el problema de mantener un estado consistente de un remate a través de una WAN. La restricción más fuerte que se impone para mantener la consistencia, es que los mensajes lleguen a todos los participantes y rematadores en el mismo orden.

La solución propuesta en primera instancia (el algoritmo de Bakery) es sencilla, pero introduce un punto centralizado que, como en todo sistema distribuido, es un punto de falla del cual depende todo el sistema. El esquema de replicación descripto anteriormente es para evitar esto. Se puede ver que el sistema depende de los brokers. Sin embargo, como corresponde a un sistema distribuido, un fallo en alguno de los brokers no hace fallar al sistema completo sino sólo a la parte afectada. Es decir que ante la falla de un bróker los remates que no están sobre este pueden continuar sin problemas.

Un punto pendiente, e importante en este tipo de sistemas, es la seguridad, no solamente desde el punto de vista del espionaje o de una intrusión, sino también por el no repudio o autenticación. Es decir que cuando se recibe una oferta se pueda asegurar que la realizó quien dice hacerlo. El trabajo no incurrió en ninguno de estos aspectos.

Implemento un prototipo del sistema para probar los dos ejes fundamentales del mismo: el orden de los mensajes y la reacción ante la caída del Distribuidor Remoto que ejecuta el rol de líder. Las pruebas hechas con el prototipo, fueron en el laboratorio y sobre una LAN montando un SRDR, tres DRs, dos DLs, tres participantes y un rematador (todos sobre distintos equipos). Las pruebas de orden de mensajes las hice sincronizando manualmente dos participantes para enviar ofertas y las pruebas de recuperación ante la caída del líder, haciendo caer el DR correspondiente.

Ambos tipos de pruebas realizadas con el prototipo tuvieron resultados favorables. La aleatoriedad en el envío de mensajes usando la capa de transporte hace extremadamente difícil controlar el ambiente de prueba, así como el hecho de estar trabajando sobre una LAN hace que no se puedan probar situaciones de demoras en la entrega de mensajes entre brokers que sí aparecen sobre una WAN. Sin embargo, repitiendo la misma prueba varias veces pude obtener al menos un caso que pusiera de relieve la problemática estudiada.

A partir de esta plataforma, el desarrollo de un sistema para remate resulta muy sencillo. Esto se debe a que la plataforma oculta todas las problemáticas del intercambio de mensajes a la aplicación, y esta sólo entiende de las funcionalidades

propias del remate. Como resultado de esto, el programador se ocupa solamente de la lógica de negocio.

References

1. P. Verissimo and L. Rodrigues. *Distributed Systems for System Architects*, Ed. Kluwer Academic Publishers, 2001.
2. Andrew S. Tanenbaum and Marteen van Steen. *Distributed Systems: Principles and Paradigms*, Ed. Prentice Hall, 2002.
3. N. Lynch. *Distributed Algorithms*, Ed. Morgan Kaufmann, 1996.
4. Leslie Lamport. *Time, clocks and the ordering of events in a distributed system*, Communications of the ACM, 1978.
5. Mojtaba Hosseini, Dewan Tanvir Ahmed, Shervin Shirmohammadi, and Nicolas D. Georganas. *A Survey of Application-Layer Multicast Protocols*, IEEE Communication Surveys and Tutorials, 2007.
6. P. Maheshwari, H. Tang, R. Liang. *Enhancing Web Services with Message-Oriented Middleware*, Proceedings of the IEEE International Conference on Web Services, 2004.
7. Hubert Zimmermann. *OSI Reference Model. The ISO Model of Architecture for Open Systems Interconnection*, IEEE Transactions on Communications, 1980.
8. R. van Renesse, T. M. Hickey and K. P. Birman. *Design and performance of Hours: A lightweight group communications system*, Tech. Rep. 94-1442, Cornell University, Department of Computer Science, 1994.
9. L. Gilman and R. Schreiber. *Distributed Computing with IBM MQSeries*, Wiley, 1996.
10. M. Hapner, R. Burridge and R. Sharma. *Java Message Service Specification*, Technical report, Sun Microsystems, http://java.sun.com/products/jms, Nov. 1999.