



**Instituto Politécnico Nacional**  
Escuela Superior de Cómputo



Desarrollo de Sistemas Distribuidos  
Prof. **Benjamín Cruz Torres**

## **Práctica No. 6**

### **Sincronización usando relojes lógicos**

**Grupo: 4CV3**

**Equipo: 6**

Integrantes:

1. Acosta Rosales Jair Sebastián
2. De Jesús López David
3. Galicia Vargas Gerardo
4. Martínez Marcos Luis Eduardo
5. Octaviano Lima Elvia Jaqueline

*Fecha: 09 de Noviembre 2018*

## Práctica 6: Sincronización usando relojes lógicos

Objetivo de la Práctica Que el alumno comprenda el funcionamiento y las ventajas de utilizar la sincronización con relojes lógicos.

Tecnologías a aplicar: Sockets, RMI, SOAP, Hilos (threads), POO, Protocolos de comunicación, Bases de Datos, algoritmos de sincronización.

### Actividades

A partir de la práctica 4, desarrollará una aplicación para repartir cartas de baraja francesa a través de dos servidores. Los servidores deberán sincronizar sus relojes lógicos. De acuerdo a los siguientes requerimientos:

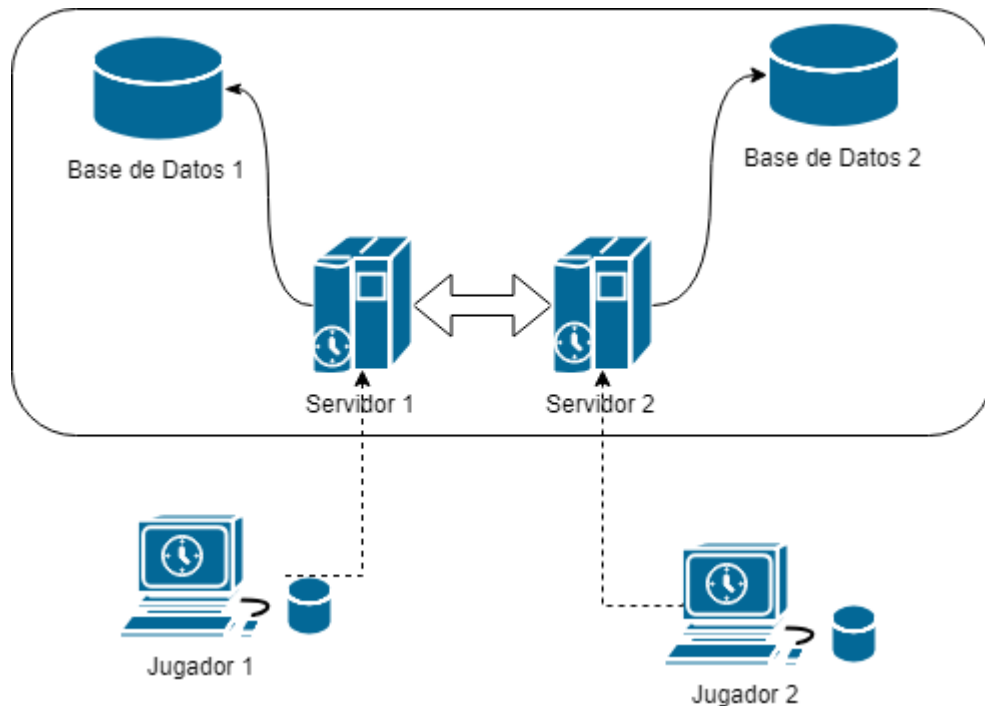


Figura 1. Sincronización usando relojes lógicos

### Requerimientos funcionales

- Los servidores son los únicos conectados y con acceso a las bases de datos.
- Dentro de la interfaz de cada servidor repartidor habrá un botón de "Reiniciar" y un Canvas para poner una imagen, inicialmente vacío.
- En cada computadora Servidor y Jugador incluirá un reloj lógico.
- En cada computadora Jugador hay un botón de "Pedir carta". Éste manda una petición al servidor correspondiente, con la cual dicho servidor envía la información de una carta al azar a ese Jugador.
- La información de petición (IP, hora, carta) se guardará en la base de datos.
- El botón "Reiniciar", permitirá reiniciar la partida del lado de cada servidor.
- La carta elegida se mostrará (en forma de imagen) solamente en la interfaz gráfica del coordinador. En el cliente se mostrará solamente en formato de texto.
- Cuando termina la partida (se repartieron todas las cartas) se le notificará al usuario si quiere salir o reiniciar una nueva partida.

## Requerimientos no funcionales

- El cliente 1 será atendido siempre por el servidor 1, el cliente 2 será atendido siempre por el servidor 2.
- Para sincronizar se utilizará el algoritmo de Lamport para relojes lógicos.
- Los equipos sincronizarán sus relojes cada que se envíen un mensaje entre ellos.
- Los relojes lógicos incrementarán su valor en 1 cada segundo.

## Investigación

La primer parte de la investigación requirió saber como se iba a hacer la conexión entre ambos servidores para que se enviarán sus respectivos contadores, llegando a lo que se conoce como callbacks, de modo que esto nos permite hacer una comunicación bidireccional entre lo que es un servidor y un cliente, para nuestro caso uno de los servidores funcionara como cliente del otro pero al mismo tiempo contendrá una referencia al otro servidor para que de este modo ambos estén conectados.

## Introducción

Usando lo que se conoce como Invocación de Métodos Remota (Remote Method Invocation), permitiendo el uso de métodos externos en una forma tal que pareciera fueran internos se realizará una práctica que permita conectar dos servidores, cada uno co su base de datos respectiva que almacenará tanto las cartas entregadas al jugador al que atienden como las cartas entregadas al jugador del otro servidor, siendo necesario que ambos servidores se conozcan en el entorno.

## Desarrollo de la práctica

Considerando el desarrollo de las anteriores prácticas, se partió del punto en que ya se contaba con la funcionalidad de los relojes y la comunicación entre los clientes y los servidores.

Para la comunicación entre servidores se hizo uso de algo conocido como callbacks, de modo pudiéramos conectar dos servidores y ambos hicieran inserts remotos a la base de datos del otro servidor, además se hizo uso de un contador lógico que, con la ayuda de la implementación del algoritmo de Lamport fue de utilidad para llevar una correcta sincronización entre los eventos que pasaban uno después del otro entre ambos servidores.

Para la persistencia se consideró un modelo de base datos con las tablas necesarias para almacenar la información justa sobre las partidas, las cartas y la asignación de estas a los jugadores. El modelo que se realizó es el mostrado a continuación:

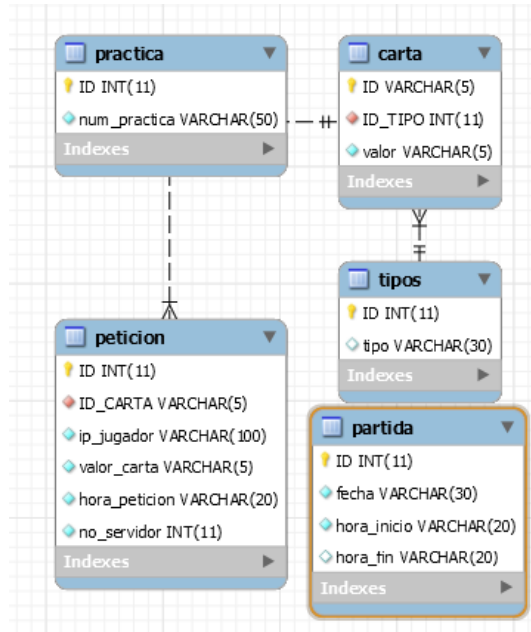


Figura 3. Modelo de base de datos

Los modelos de las bases de datos se poblaron con las rutas de las cartas, cuyas imágenes se encuentran almacenadas en las máquinas donde se ejecutan los servidores. Así, pues el Servidor1 está asociado al modelo *juegocartass1*, mientras que el Sevidor2 se asoció con *juegocartass2*.

Posteriormente, se revisó el funcionamiento del mecanismo RMI, del cual se observó que es provisto por el entorno de ejecución de Java, por lo que una de sus principales limitaciones es que es exclusivo para su utilización con tal entorno de ejecución.

Por otro lado, se observó que consiste en un objeto cliente que realiza un requerimiento de datos, desde uno de sus objetos locales, para después enviarlo a un objeto ubicado en un servidor. El objeto en el servidor realiza las operaciones correspondientes para preparar la información requerida. Finalmente, el objeto en el servidor envía la respuesta al cliente.

Primero se procedió a definir las operaciones de cada cliente y servidor, cuyas firmas se tenían que registrar en la correspondiente interfaz, la cual es la base fundamental de que los objetos mantuvieran una comunicación remota.

Luego, se definieron los detalles de las operaciones, donde los clientes esencialmente solicitan cartas a los servidores (además de una prueba adicional que se incluyó para verificar el funcionamiento, en la que un cliente recibe la consulta que remotamente aplica el servidor sobre su base de datos asociada), mientras que cada servidor agrega al otro y cada uno puede realizar transacciones sobre sus bases de datos asociadas.

La función de reloj en cada cliente y servidor se mantuvo como en las prácticas anteriores, donde cada uno tiene su propio reloj con una hora modificable y es tal hora la que utilizan los servidores para almacenar

Puesto que cada servidor agrega al otro, uno de ellos(en este caso el servidor 2) debe incluir la dirección IP del otro, no obstante, por el orden en que se ejecutan las operaciones, se debe iniciar primero el Servidor1 y con la ayuda de una función remota en cada interfaz de servidor podemos enviar el reloj lógico de aquel que ejecuta la función y compararlo con el reloj lógico del otro servidor, de modo que con esto se puede hacer fácilmente la ejecución del algoritmo de lamport. Una vez iniciados los servidores, los clientes pueden comenzar a solicitar cartas. Tras la primera solicitud de una carta se inicia una nueva partida. Cada servidor puede reiniciar la partida. Todas las operaciones de partidas y cartas asignadas a jugadores se van registrando en la base de datos.

En la imagen se muestran las interfaces de cada uno de los servidores, con su correspondiente canvas, donde se muestra la imagen de la carta asignada al jugador asociado, y las operaciones que puede realizar. Por su parte, las interfaces de los jugadores muestran el nombre de las cartas recibidas y los botones para sus correspondientes funcionalidades.

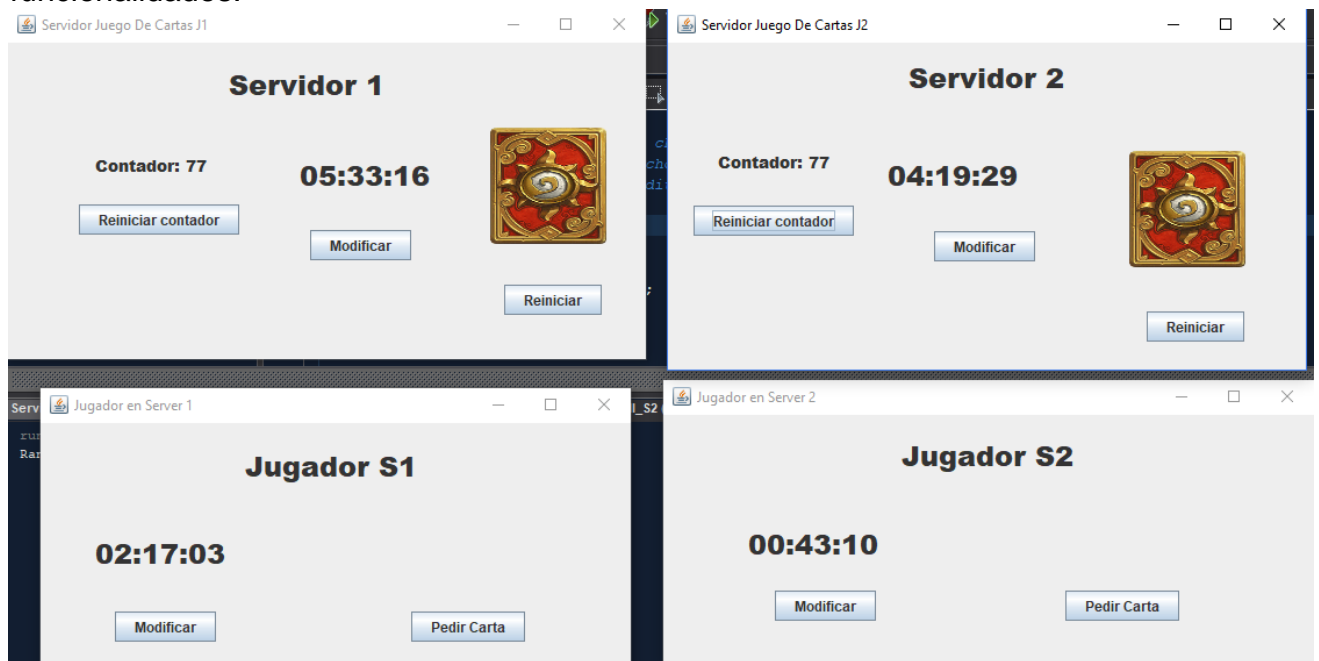


Figura 4. Interfaces de clientes y servidores.

A continuación se muestran las pruebas realizadas.

Se ejecutan los servidores.



Cada interfaz de servidor se inicializa con un reloj, cuya hora se genera aleatoriamente. En la consola se escriben las conexiones a base de datos y los resultados de las operaciones, además contienen el reloj lógico que se actualiza cada que alguno ejecuta una llamada remota del otro servidor, en la imagen se puede ver que ya están sincronizados.

```

el contador es correcto en S1
nombreCarta: 3 de Corazones Clave: 13
idCarta: 13 ipJugador: Sebastián-AR/192.168.56.1 valorCarta: 3 horaPetición: 05:34:15
Fri Nov 02 23:40:52 CST 2018 WARN: Establishing SSL connection without server's identity
Exito al guardar los datos en la bd(insertPetición)
Fri Nov 02 23:40:56 CST 2018 WARN: Establishing SSL connection without server's identity
Consulta Remota efectuada en Servidor 1
Hice actualización del contador en S1
12
14
15
Eliminare la carta: 15
12
14
el contador es correcto en S1

```

```

Consulta Remota efectuada en Servidor 2
Hice actualización del contador en S2
12
13
14
15
12
14
15
el contador es correcto en S2
nombreCarta: 5 de Corazones Clave: 15
idCarta: 15 ipJugador: Sebastián-AR/192.168.56.1 valorCarta: 5 horaPetición: 04:20:32
Fri Nov 02 23:40:56 CST 2018 WARN: Establishing SSL connection without server's identity
Exito al guardar los datos en la bd(insertPetición)
Conexion terminada..

```

En la base de datos se registra la partida recién iniciada.

55	11	Sebastián-AR/192.168.56.1	As	04:20:24	2
56	13	Sebastián-AR/192.168.56.1	3	05:34:15	1
57	15	Sebastián-AR/192.168.56.1	5	04:20:32	2

Se ejecutan los clientes, cada uno corresponde a un jugador asociado a un servidor.




El Jugador1 solicita una carta, el Servidor1 elije aleatoriamente una de las cartas y realiza la correspondiente transacción para guardar la solicitud de carta, el Jugador1 recibe la carta. El Servidor2 obtiene la misma sentencia para persistir la asignación de la carta en su base de datos asociada, con ayuda de la última columna podemos saber que servidor hizo cada consulta.

**Servidor 1**

Contador: 692  
05:43:28

Reiniciar contador  
Modificar



Reiniciar

Jugador en Server 1

**Jugador S1**

02:27:14  
3 de Corazones

Modificar  
Pedir Carta

```
mysql> SELECT * FROM peticion;
```


ID	ID_CARTA	ip_jugador	valor_carta	hora_peticon	no_servidor
1	11	Sebastián-AR/192.168.56.1	As	17:28:37	2
2	14	Sebastián-AR/192.168.56.1	4	06:47:48	1
3	13	Sebastián-AR/192.168.56.1	3	06:47:55	1
4	15	Sebastián-AR/192.168.56.1	5	17:29:10	2
5	12	Sebastián-AR/192.168.56.1	2	17:29:15	2
6	11	Sebastián-AR/192.168.56.1	As	06:48:25	1
7	15	Sebastián-AR/192.168.56.1	5	14:04:54	2
8	13	Sebastián-AR/192.168.56.1	3	14:05:03	2
9	12	Sebastián-AR/192.168.56.1	2	06:48:48	2
10	14	Sebastián-AR/192.168.56.1	4	14:05:21	2
11	15	Sebastián-AR/192.168.56.1	5	16:34:47	1
12	13	Sebastián-AR/192.168.56.1	3	04:10:04	2
13	14	Sebastián-AR/192.168.56.1	4	16:35:21	1
14	12	Sebastián-AR/192.168.56.1	2	04:10:36	2
15	11	Sebastián-AR/192.168.56.1	As	04:10:44	2
16	32	Sebastián-AR/192.168.56.1	2	04:11:04	2
17	213	Sebastián-AR/192.168.56.1	K	04:11:24	2
18	48	Sebastián-AR/192.168.56.1	8	16:36:28	1
19	28	Sebastián-AR/192.168.56.1	8	04:11:45	2
20	25	Sebastián-AR/192.168.56.1	5	04:11:53	2
21	113	Sebastián-AR/192.168.56.1	K	04:11:56	2
22	311	Sebastián-AR/192.168.56.1	J	16:36:59	1
23	44	Sebastián-AR/192.168.56.1	4	16:37:07	1
24	411	Sebastián-AR/192.168.56.1	J	04:12:16	2
25	110	Sebastián-AR/192.168.56.1	10	16:37:21	1

Se repite la operación, esta vez para el Jugador2 y Servidor2. Esta vez se observan las dos solicitudes de cartas y se ven ambas solicitudes reflejadas en las bases de datos, tanto del Servidor1 como del Servidor2.

**Servidor 2**

Contador: 789  
04:31:19

Reiniciar contador  
Modificar



Reiniciar

Jugador en Server 2

**Jugador S2**

00:55:00  
5 de Corazones

Modificar  
Pedir Carta

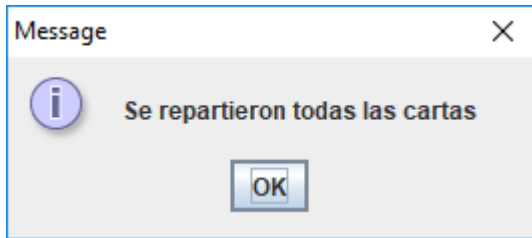
```
mysql> SELECT * FROM peticion;
```

ID	ID_CARTA	ip_jugador	valor_carta	hora_peticon	no_servidor
1	11	Sebastián-AR/192.168.56.1	As	17:28:37	2
2	14	Sebastián-AR/192.168.56.1	4	06:47:48	1
3	13	Sebastián-AR/192.168.56.1	3	06:47:55	1
4	15	Sebastián-AR/192.168.56.1	5	17:29:10	2
5	12	Sebastián-AR/192.168.56.1	2	17:29:15	2
6	11	Sebastián-AR/192.168.56.1	As	06:48:25	1
7	15	Sebastián-AR/192.168.56.1	5	14:04:54	2
8	13	Sebastián-AR/192.168.56.1	3	14:05:03	2
9	12	Sebastián-AR/192.168.56.1	2	06:48:48	1
10	14	Sebastián-AR/192.168.56.1	4	14:05:21	2
11	15	Sebastián-AR/192.168.56.1	5	16:34:47	1
12	13	Sebastián-AR/192.168.56.1	3	04:10:04	2
13	14	Sebastián-AR/192.168.56.1	4	16:35:21	1
14	12	Sebastián-AR/192.168.56.1	2	04:10:36	2
15	11	Sebastián-AR/192.168.56.1	As	04:10:44	2
16	32	Sebastián-AR/192.168.56.1	2	04:11:04	2
17	213	Sebastián-AR/192.168.56.1	K	04:11:24	2
18	48	Sebastián-AR/192.168.56.1	8	16:36:28	1
19	28	Sebastián-AR/192.168.56.1	8	04:11:45	2
20	25	Sebastián-AR/192.168.56.1	5	04:11:53	2
21	113	Sebastián-AR/192.168.56.1	K	04:11:56	2
22	311	Sebastián-AR/192.168.56.1	J	16:36:59	1
23	44	Sebastián-AR/192.168.56.1	4	16:37:07	1
24	411	Sebastián-AR/192.168.56.1	J	04:12:16	2
25	110	Sebastián-AR/192.168.56.1	10	16:37:21	1
26	45	Sebastián-AR/192.168.56.1	5	04:12:30	2
27	39	Sebastián-AR/192.168.56.1	9	16:37:41	1



Se continúa solicitando cartas hasta que se agotan las disponibles por partida. En la base de datos se observan las peticiones y el servidor que ha asignado cada carta.

Al agotar las cartas disponibles, se lanza un mensaje.



En la base de datos se registra el fin de partida.

ID	fecha	hora_inicio	hora_fin
1	26/Oct/2018	07:52:56	NULL
2	26/Oct/2018	06:40:26	NULL
3	26/Oct/2018	17:28:04	17:29:15
4	26/Oct/2018	06:46:58	06:48:10
5	26/Oct/2018	06:47:43	06:48:59
6	26/Oct/2018	14:04:05	14:05:21

Nota: dadas las pruebas hechas, en la imagen se aprecia que algunas partidas no tienen la hora final, pues estas fueron terminadas antes de que oficialmente la partida pudiera terminar repartiendo todas las cartas.

## Conclusiones

Gracias a la implementación de callbacks fue posible hacer la conexión entre ambos servidores, aunque originalmente este método es implementado para que un servidor envíe información a un cliente, es decir que se realice una conexión bidireccional, en este caso podemos decir que el servidor 2 funciona como cliente del server 1 de modo que pueden enviarse información entre si.

Si hacemos una comparación entre algoritmo de Cristian y Lamport podemos decir que este último es mucho más fácil de implementar dado que no hace uso de un servidor extra del cual los dos servidores de juego deben estar conectados, sin embargo es muchísimo más eficiente si lo que queremos tener es una hora única entre todos los servidores, a diferencia de Lamport que lo único que importa es saber cuál evento sucedió primero.

## Bibliografía

*Comunicación entre un servidor y múltiples clientes.* (s.f.). Webtutoriales.com. Recuperado de: <http://www.webtutoriales.com/articulos/comunicacion-entre-un-servidor-y-multiples-clientes>

González-J., Agustín. (2009). *Remote Method Invocation (Invocación Remota de Métodos)*. Recuperado de: <http://profesores.elo.utfsm.cl/~agv/elo330/2s09/lectures/RMI/RMI.html>

Alejandro Calderón Mateos, Javier García Blas, David Expósito Singh, Laura Prada Camacho,(2012). JAVA RMI: CALLBACK DE CLIENTE  
Recuperado de: <http://ocw.uc3m.es/ingenieria-informatica/desarrollo-de-aplicaciones-distribuidas/materiales-de-clase/rmi-callback>