Práctica Final: Marcador deportivo simultáneo

Sistemas Distribuidos.
Grado en Ing. en Telemática, 2019-2020.
Escuela Técnica Superior de Ingeniería de Telecomunicación.
Universidad Rey Juan Carlos.

Realización: **Individual o por parejas**.
Fecha límite de entrega: **00:00 del miércoles, 15 de enero de 2020**

---

1. Objetivos

- Implementación de un algoritmo de coherencia de cachés en entornos distribuidos.
- Comprender mejor el protocolo de *invalidación de escritura* para coherencia en cachés, así como el protocolo de escritura postergada (*write back*).
- Evaluar el rendimiento de este tipo de sistemas en la práctica, mediante simulación.

2. Enunciado

Considera un sistema de información que da soporte a un servicio de *marcador simultáneo* de eventos deportivos. En este caso, vamos a centrar el problema en un marcador simultáneo de partidos de fútbol, en el que hay una aplicación cliente que se encuentra en cada estadio, siguiendo el partido y actualizando el resultado del mismo. El resto de aplicaciones cliente, en otros estadios, deben tener información sobre el resultado de un estadio diferente, cuando así lo requieran. Del mismo modo, el sistema centralizado de seguimiento de resultados debe almacenar la copia maestra de todos los marcadores, al concluir los partidos.

- La puntuación máxima que se puede obtener al implementar cada apartado aparece claramente reflejada al comienzo de la sección correspondiente.
- Se entregará una sola versión de la solución que incluya, simultáneamente, la solución a todos los apartados que se hayan resuelto, implementada en Go.
- No es obligatorio resolver todos los apartados, ni tampoco resolver las mejoras sobre la práctica básica en el orden indicado en el enunciado.
- Para superar la práctica, el alumno debe obtener como mínimo una nota de 5 puntos sobre 10.

3. Práctica básica: protocolo de invalidación de escritura

Puntuación: 5 puntos

Consideramos que cada cliente en un estadio será una gorutina en Go. Tendremos en total 4 estadios, $\{E_0, \ldots, E_3\}$, con 4 partidos celebrándose de forma simultánea en cada uno de ellos.

Para no alargar en exceso la ejecución del programa, la duración de todos los partidos será de 20 segundos. Todos deben comenzar a la vez.

La gorutina que simula el transcurso de cada partido debe decidir, (supongamos que aleatoriamente), si uno de los dos equipos mete un gol. En ese caso, tendrá que actualizar su propio marcador. para ello, en cada partido:

- Cada 2 segundos, en el partido se ejecuta una rutina que simula una ocasión de gol (no hay más eventos posibles en esta versión básica).
- En esta rutina, se saca un número aleatorio en el intervalo $[0, 1]$. Si el número es menor o igual a 0.5 la ocasión es del equipo local, en caso contrario se asigna al equipo visitante.
- A continuación, la rutina saca otro número aleatorio en el intervalo $[0, 1]$. Si el número es menor o igual a 0.7 la ocasión es fallida. En caso contrario, el equipo seleccionado en el paso anterior marca gol.
- En consecuencia, la rutina se ejecuta 10 veces durante el transcurso del partido completo para simular el resultado final del mismo.
- **Importante**: cada vez que un equipo marque un gol en uno de los estadios, se debe actualizar el valor del marcador para ese estadio y, a continuación, **invalidar** la copia del resultado en cualquier otro estadio que siga ese marcador.

Cada 5 segundos, el *sistema central* pedirá una ronda informativa a todos los estadios para conocer su versión del resultado. A fin de practicar el protocolo de mantenimiento de coherencia entre cachés, los estadios tendrán que mantener información de su propio partido y de otro(s) partido(s) de interés, de la siguiente forma:

- Estadio 1: Resultado propio y del estadio 3.
- Estadio 2: Resultado propio y de los estadios 1 y 3.
- Estadio 3: Resultado propio y del estadio 4.
- Estadio 4: Resultado propio y del estadio 2.

La dinámica de implementación de la ronda será la siguiente:

- El sistema central elige al azar uno de los 4 estadios para empezar la ronda. Los 4 estadios deben tener la misma probabilidad (aproximadamente) de ser elegidos. Una vez elegido el estadio inicial, la ronda continúa por los demás estadios en orden, siguiendo un buffer circular. Es decir, si por ejemplo empezamos por el estadio E_2, el orden de consulta será E_2, E_3, E_0, E_1.
- En cada estadio, se debe enviar el resultado propio y el del estadio o estadios de interés que también se siguen en ese partido. Si el resultado es **inválido** (porque no esté actualizado), se tendrá que llevar el resultado válido de la caché local correspondiente.
- Si el servicio central no tiene el resultado actualizado, primero actualizo al servicio central (*write back*), y luego al estadio que necesita el resultado.
- Si el servicio central sí que tiene el resultado actualizado, accedo al servicio central para que me lo dé.
- Al final de todos los partidos, todas las cachés locales (de cada estadio) y el servicio central deberían tener copias válidas de todos los resultados.

Al final de la simulación, el programa debe imprimir por pantalla:

- Resultado final del partido en cada estadio, así como de los partidos de interés en cada estadio.

- Resultado final registrado por el sistema central.
- Número de operaciones de *write back* que hemos realizado en total.

Sugerencias de implementación

- Como mínimo, debe haber una gorutina por cada estadio, otra gorutina para el servicio central y otra más para el protocolo de coherencia entre cachés y con la memoria principal.
- Para no saturar la consola con muchos mensajes, y facilitar la auditoría de ejecución del programa, se aconseja encarecidamente que **todos los mensajes de progreso** que contengan trazas de estado y ejecución (e.g. goles marcados, acciones de los clientes, del sistema de control de coherencia, etc.) se escriban **en un ficher log**, utilizando el módulo estándar correspondiente de Go.

4. Adicional 1: Protocolo *snoopy*

Puntuación: 3 puntos

En esta mejora, se modifica la implementación anterior para que, cuando un estadio no tenga el resultado de otro estadio de interés en ese partido, solicite directamente la información al estadio de origen, a través de un canal de Go, en lugar de recuperarla del servicio central.

- Cada estadio tendrá un solo canal centralizado para recibir solicitudes de los demás estadios.
- Las solicitudes de actualización se procesarán por orden de llegada.

Calcula cuántas peticiones de actualización tiene que servir cada estadio al resto. Si esto fuese un sistema de coherencia entre cachés, y en el paso directo de mensajes entre cachés tardase la mitad de tiempo por petición que al actualizar los datos de caché a memoria principal (el "servicio central" en nuestro simulador): ¿qué estrategia crees que es más efectiva? ¿Por qué?

5. Adicional 2: Estadísticas de uso

Puntuación: 2.5 puntos

Añade funcionalidad al sistema de control de coherencia para que cuente cuántas veces tiene que invalidar una caché (o varias) que tengan copias de ese contenido (el resultado varía en cada ejecución, en función de los goles marcados en ese partido).

Obtén el promedio de todas las estadísticas anteriores y de esta métrica nueva del párrafo anterior para cinco simulaciones consecutivas del programa. Considera que (u.t. es "unidad de tiempo", en genérico):

- en invalidar otra línea de una caché diferente tardo 0.5 u.t.;
- en actualizar esa línea en otra caché diferente tardo 1 u.t.;
- en actualizar la memoria principal con el nuevo valor de una línea tardo 2 u.t.

¿Puedes justificar, con datos objetivos (a partir de las simulaciones) por qué una estrategia de *write back* + invalidación es más eficiente que una de *write through* con propagación de cambios?