Distribución de datos y *Map Reduce*

PEC2

Ejercicio 1 (25%)

La Bolsa de Madrid ha encargado a una empresa crear una aplicación que permita a los usuarios registrados consultar los datos bursátiles que se van generando en tiempo real durante la actividad diaria de la bolsa así como de otras bolsas de todo el mundo. En cualquiera de los casos con un retardo de 15 minutos. Así mismo, la aplicación ofrecerá una API REST de manera que empresas financieras registradas puedan crear aplicaciones con servicios de valor añadido para sus clientes a partir de los datos que se recuperen usando los servicios de la API. Se sabe que el número de datos que se generan en cada instante de tiempo son enormes, y que el número de consultas de la información que se van a producir concurrentemente también será enorme.

1. **MODELO**. No existen relaciones relevantes que deban priorizarse a la hora de almacenar datos bursátiles, por lo que se puede prescindir del modelo relacional en este caso.
   Por lo tanto, un modelo **NoSQL** sería lo más conveniente: permite escalar el volumen y el acceso de los datos de forma más eficiente. Además, con un sistema distribuido, su disponibilidad es mayor que con un sistema centralizado en el que la concurrencia multiplica la complejidad del sistema, ocasionando problemas de rendimiento que afectarían directamente a los usuarios.
   No obstante, la arquitecta del sistema distribuido puede propiciar cuellos de botella si no se diseña acorde a las necesidades de la aplicación. Un **sistema de 3 capas** evitaría que la carga lógica recayera sobre los usuarios -además de evitar la saturación de la red, dado que en este caso se esperan grandes volúmenes de consultas- o que el rendimiento y la disponibilidad se vean comprometidos, como podría ocurrir con cualquier configuración de un sistema cliente/servidor; estos inconvenientes se solventarían aplicando una capa de negocio intermedia que gestionase tanto las peticiones de los clientes como la gestión de los datos de los servidores.
   Por otro lado, un sistema P2P sería demasiado descentralizado en este caso, ya que no tendría sentido que los datos bursátiles se almacenasen en los sistemas de los usuarios -lo que implicaría, además, cierta volatilidad-; en este sentido, es necesaria cierta centralización, marcando la diferencia entre un cliente y un servidor.

2. **FRAGMENTACIÓN**. Dado que los datos albergan información bursátil de distintas bolsas del mundo, es lógico pensar que los clientes se interesarán en aquellos registros en los que operen en mayor medida, ignorando aquellos que les aporten valor. Debido a esta selección de los datos, la fragmentación **horizontal** encajaría perfectamente, pudiendo realizar la fragmentación por país, bolsa o, afinando aún más, por mercado o sector.
   En cuanto a una fragmentación vertical, no sería necesaria en caso de almacenar exclusivamente los datos -columnas- de valor para los usuarios. Siendo así, carecería de sentido fragmentar las columnas ya que los clientes necesitarían acceder a todas ellas -teniendo también en cuenta la homogeneidad de los perfiles de éstos-. Por lo tanto, se pueden descartar la fragmentación vertical e híbrida.

3. **REPLICACIÓN**. Dado que un usuario que acceda de forma intensiva a los datos bursátiles de, por ejemplo, la Bolsa de Nueva York puede estar físicamente en cualquier parte del mundo de manera habitual y sin que se considere una anomalía, considerar la replicación horizontal de datos -***sharding***- en lugar de la fragmentación sería del todo acertado -al menos para aquellos fragmentos que se esperen sean accedidos repetidamente desde distintos nodos-. Asegurar la consistencia será, por tanto, de vital importancia. Entre otros, una gestión de réplicas ***Master-Slave*** de **consistencia síncrona** sería la que mejor se adaptaría a este caso, ya que tan solo se escriben en los nodos primarios -que almacenan los fragmentos a ser actualizados-, y a partir de ahí se propagaría hacia los secundarios, asegurando la consistencia durante el proceso.

4. **TRANSACCIONES**. Esta no es una decisión trivial dada la concurrencia de los usuarios y la alta frecuencia de actualización de los datos, circunstancias que pueden comprometer la consistencia. Si la gestión de réplicas se implementa correctamente, se puede sacrificar la consistencia en algunos casos: si suponemos que la replicación se ha hecho en base a las distintas bolsas del mundo, no es del todo crítico que las réplicas de la Bolsa de Madrid y las de la Bolsa de Nueva York tengan un *timestamp* diferente -esto es, no se han actualizado al mismo tiempo- dado que los usuarios accederán mayormente a los datos de una u otra. Lo que sí sería crítico es asegurar la consistencia entre las réplicas de las mismas bolsas en este caso, pero no la consistencia entre todas ellas. Por ejemplo, si los datos de la Bolsa de Nueva York no están disponibles por alguna razón, no sería lógico que los usuarios no pudieran acceder a los datos de la Bolsa de Madrid. Por lo tanto, el modelo **BASE** es el encajaría mejor con estos requerimientos sin sacrificar la disponibilidad.

Ejercicio 2 (25%)

A partir de la lectura de los apuntes de la asignatura, indicad qué os parecen las siguientes afirmaciones. Para cada una de las afirmaciones:

- Indicad si es cierta o falsa. No serán válidas las respuestas que no indiquen si la afirmación es cierta o es falsa.
- Justificar brevemente vuestra respuesta haciendo referencia al libro o los apuntes de la asignatura.

Afirmación 1

La fragmentación horizontal significa que la eficiencia de la base de datos puede crecer simplemente añadiendo más nodos a la red de ordenadores.

VERDADERO. La fragmentación horizontal significa que se distribuyen subconjuntos de objetos que comparten semántica a partir del valor de uno o varios atributos, lo que permite incrementar la eficiencia del sistema añadiendo más nodos -escalando horizontalmente-.

Página 8 de la locución del vídeo del tema 5: *Bases de datos distribuidas: Diseño*. ([1])

Afirmación 2

En un SGDB relacional, para hacer una operación, además de conectarse a la base de datos es imprescindible tener una transacción activa (o en ejecución), que siempre es única.

VERDADERO. Un SGBD relacional debe establecer y mantener una conexión -sesión de trabajo- con la base de datos, además de tener una transacción activa única iniciada manual o automáticamente al realizar la primera operación.

Página 10 del módulo *Gestión de Transacciones*. ([2])

Afirmación 3

El data sharding (poner diferentes partes de los datos en diferentes servidores), proporciona una forma de escalar horizontalmente solo las operaciones de lectura.

FALSO. El data sharding proporciona una forma de escalar horizontalmente cualquier tipo de operación con un sistema de control de concurrencia y consistencia.

Páginas 13 y 14 de la locución del vídeo del tema 5: *Bases de datos distribuidas: Diseño*. ([1])

Afirmación 4

La replicación de datos en un clúster ayuda a evitar conflictos generados por actualizaciones simultáneas.

FALSO. Al realizar actualizaciones simultáneas sobre dos réplicas, éstas se actualizan mutuamente. En caso de que la nueva información no sea idéntica, se genera un conflicto que provoca inconsistencia, dado que cada réplica se actualiza con el valor de la otra réplica.

Páginas 14 de la locución del vídeo del tema 6: *Bases de datos distribuidas: modelo ACID*. ([3])

Ejercicio 3 (30%)

Considerar el producto de las siguientes matrices: **[[2, 3, 6], [1, 0, 3], [3, 1, 4]]**

M1: $\begin{pmatrix} 2 & 3 & 4 \\ 1 & 0 & 2 \\ 3 & 1 & 1 \end{pmatrix}$ y M2: $\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Se va a implementar el producto usando 2 pasadas map-reduce. Para ello se supondrá que las matrices se representan mediante una lista con tuplas de la forma (Mi,i,j,v) donde:

- Mi representa el nombre de la matriz
- i representa el índice de una fila de Mi
- j representa el índice de una columna de Mi
- v representa el valor de la posición (i,j) en la matriz Mi

Por tanto, la lista de entrada sería:
[(M1,1,1,2), (M1,1,2,3), (M1,1,3,4), (M1,2,1,1), (M1,2,2,0), (M1,2,3,2), (M1,3,1,3), (M1,3,2,1), (M1,3,3,1), (M2,1,1,1), (M2,1,2,0), (M2,1,3,1), (M2,2,1,0), (M2,2,2,1), (M2,2,3,0), (M2,3,1,0), (M2,3,2,0), (M2,3,3,1)]

Y el resultado debería ser tuplas de la forma ((i,j),v), donde v representa el valor de la fila i y la columna j de la matriz producto. Es decir el resultado sería:
[((1,1),2),((1,2),3),((1,3),6),((2,1),1),((2,2),0),((2,3),3),((3,1),3),((3,2),1),((3,3),4)]

Se pide explicar en cada pasada map-reduce lo que ocurre en cada fase (map, Shuffle y reduce). Concretamente:

- Explicar qué acciones se producen.
- Mostrar los datos de entrada
- Mostrar el resultado producido por esas acciones en los datos de entrada

No es necesario codificar las acciones que se producen en cada fase, pero sí explicar claramente las acciones realizadas.

PASADA 1

MAP

Entrada:
[(M1,1,1,2), (M1,1,2,3), (M1,1,3,4), (M1,2,1,1), (M1,2,2,0), (M1,2,3,2), (M1,3,1,3), (M1,3,2,1), (M1,3,3,1), (M2,1,1,1), (M2,1,2,0), (M2,1,3,1), (M2,2,1,0), (M2,2,2,1), (M2,2,3,0), (M2,3,1,0), (M2,3,2,0), (M2,3,3,1)]

En esta primera fase se recorre la lista anterior y, para cada tupla (*Mi*,*i*,*j*,*v*) se establecen dos claves: (***Mi***) e (***i***,***j***); por lo que cada tupla de la lista será de la forma ((***Mi***),(***i***,***j***),*v*).

Salida:

[((**M1**),(**1**,**1**),2), ((**M1**),(**1**,**2**),3), ((**M1**),(**1**,**3**),4), ((**M1**),(**2**,**1**),1), ((**M1**),(**2**,**2**),0), ((**M1**),(**2**,**3**),2), ((**M1**),(**3**,**1**),3), ((**M1**),(**3**,**2**),1), ((**M1**),(**3**,**3**),1), ((**M2**),(**1**,**1**),1), ((**M2**),(**1**,**2**),0), ((**M2**),(**1**,**3**),1), ((**M2**),(**2**,**1**),0), ((**M2**),(**2**,**2**),1), ((**M2**),(**2**,**3**),0), ((**M2**),(**3**,**1**),0), ((**M2**),(**3**,**2**),0), ((**M2**),(**3**,**3**),1)]

Shuffle & Sort

Utilizando como entrada la lista anterior, esta función agrupa las tuplas en función de la clave (***Mi***), resultando en una lista de forma [((***Mi***), [sub-lista]), ((***Mi***), [sub-lista])], conteniendo cada una de las sub-listas internas las tuplas correspondientes a cada matriz con la forma ((***i***,***j***),*v*).

Salida:
[((**M1**), [((**1**,**1**),2),((**1**,**2**),3),((**1**,**3**),4),((**2**,**1**),1),((**2**,**2**),0),((**2**,**3**),2),((**3**,**1**),3),((**3**,**2**),1),((**3**,**3**),1)]),
((**M2**), [((**1**,**1**),1),((**1**,**2**),0),((**1**,**3**),1),((**2**,**1**),0),((**2**,**2**),1),((**2**,**3**),0),((**3**,**1**),0),((**3**,**2**),0),((**3**,**3**),1)])]

Reduce

Siendo la lista anterior la entrada de esta función, para la lista de tuplas de **M1** se elimina el valor ***j*** de cada tupla, resultando en ((***i***),*v*); y para la lista de tuplas de **M2** se elimina el valor ***i***, resultando en ((***j***),*v*).

Salida:
[((**M1**), [((**1**),2),((**1**),3),((**1**),4),((**2**),1),((**2**),0),((**2**),2),((**3**),3),((**3**),1),((**3**),1)]),
((**M2**), [((**1**),1),((**2**),0),((**3**),1),((**1**),0),((**2**),1),((**3**),0),((**1**),0),((**2**),0),((**3**),1)])]

PASADA 2

MAP

Entrada:
[(**M1**), [(**1**),2),(**1**),3),(**1**),4),(**2**),1),(**2**),0),(**2**),2),(**3**),3),(**3**),1),(**3**),1)]),
(**M2**), [(**1**),1),(**2**),0),(**3**),1),(**1**),0),(**2**),1),(**3**),0),(**1**),0),(**2**),0),(**3**),1)])]

En esta segunda pasada se parte de la salida de la pasada anterior, por lo que la entrada de la función *map* es la salida de la función *reduce* de la pasada 1.

Esta vez, la función recorre la lista anterior y, para cada sub-lista de ***Mi***, se crea una tupla con clave ***i*** o ***j*** y con los valores ***v*** que compartan esa misma clave ***i*** o ***j***, resultando en ((***i***),($v1_i$,$v2_i$,$v3_i$)) para las tuplas de la sub-lista de M1 y ((***j***),($v1_j$,$v2_j$,$v3_j$)) para las de la sub-lista de M2.

Salida:
[(**M1**), [(**1**),(2,3,4)), (**2**),(1,0,2)), (**3**),(3,1,1))]),
(**M2**), [(**1**),(1,0,0)), (**2**),(0,1,0)), (**3**),(1,0,1))])]

Shuffle & Sort

La función *shuffle* parte de la salida anterior y hace un *join* con las tuplas de la sub-lista de M1 con las de la sub-lista de M2, y descarta la clave ***Mi***. Además, junta las claves de las tuplas en las sub-listas. Por lo tanto, cada tupla será de la forma ((***i,j***),[sub-lista]) -recordemos que ***i*** viene de las tuplas de M1; y ***j***, de las de M2-, y cada sub-lista resultará con la forma [($v1_i$,$v2_i$,$v3_i$),($v1_j$,$v2_j$,$v3_j$)].

Salida:
[((**1,1**),[(2,3,4),(1,0,0)]), ((**1,2**),[(2,3,4),(0,1,0)]), ((**1,3**),[(2,3,4),(1,0,1)]),
((**2,1**),[(1,0,2),(1,0,0)]), ((**2,2**),[(1,0,2),(0,1,0)]), ((**2,3**),[(1,0,2),(1,0,1)]),
((**3,1**),[(3,1,1),(1,0,0)]), ((**3,2**),[(3,1,1),(0,1,0)]), ((**3,3**),[(3,1,1),(1,0,1)])]

Reduce

Por último, la función reduce utiliza como entrada la salida anterior y realiza la siguiente operación:

Para cada sub-lista de la clave (***i,j***), con la forma [($v1_i$,$v2_i$,$v3_i$),($v1_j$,$v2_j$,$v3_j$)], calcula [$v1_i$*$v1_j$ + $v2_i$*$v2_j$ + $v3_i$*$v3_j$]. El valor resultante v_f es el que queda en la tupla con forma ((***i,j***),v_f). De esta forma, la salida de la función -y de la segunda pasada *map-reduce*- es la siguiente lista:

Salida:
[((**1,1**),2),((**1,2**),3),((**1,3**),6),((**2,1**),1),((**2,2**),0),((**2,3**),3),((**3,1**),3),((**3,2**),1),((**3,3**),4)]

Ejercicio 4 (20%)

Considera los siguientes casos:

1. Un sistema con una consistencia fuerte donde las lecturas y escrituras tienen la misma importancia.
2. Un sistema tolerante a fallos y disponible donde se conoce que las lecturas serán más frecuentes que las escrituras.
3. Un sistema con una consistencia fuerte donde se reciben muchas solicitudes de escritura y relativamente pocas de lectura.

Se pide identificar qué configuraciones de los valores W, R y N son las que encajan mejor con cada caso (solo una por cada caso) y razonarlo:

C1: N=3, W=3, R=1

Como $W > N/2$ y $W + R > N$, se trata de un sistema CP de consistencia fuerte. Esto no encajaría con el caso 2, ya que se sacrifica la disponibilidad en favor de la consistencia.

Por ello, sí encajaría con los casos 3 y 1; aunque, en este último, donde se les da la misma importancia a las operaciones de lectura que a las de escritura, este sistema estaría infradimensionado con $R=1$. Por lo tanto, el **caso 3** sí encaja perfectamente ya que se les da más peso a las operaciones de escritura que de lectura.

C2: N=7, W=4, R=4

Por los mismos motivos que antes, se trata de un sistema CP de consistencia fuerte; por ello, no encaja en el caso 2.

Sin embargo, en este caso el sistema sí se ha dimensionado para dar la misma importancia tanto a las operaciones de lectura como a las de escritura, ya que $R=4$ y $W=4$. Por lo tanto, encaja perfectamente con el **caso 1**, mientras que estaría sobredimensionado para el caso 3.

C3: N=9, W=4, R=5

Como $W + R \leq N$, se trata de un sistema AP donde prima la disponibilidad en detrimento de la consistencia fuerte. Esto no significa que el sistema no sea consistente en ningún momento, sino que la consistencia se garantiza a lo largo del tiempo -*eventual consistency*-. Hasta entonces, los datos disponibles pueden ser inconsistentes, por lo que se descartan los casos 1 y 3. El **caso 2** es en el que encajaría este sistema ya que se prioriza la disponibilidad

C4: N=6, W=3, R=2

Al igual que el sistema anterior, se trata de un sistema AP. Por ello, se descartan de la misma forma los casos 1 y 3, siendo el **caso 2** el escenario donde mejor encajaría. No obstante, cabe remarcar que en el caso 2 se sabe que las lecturas son más frecuentes que las escrituras, pero este sistema da más recursos a las escrituras que a las lecturas -$W>R$-, por lo que la dimensión del sistema no sería del todo óptima para el caso.

C5: N=7, W=4, R=5

Este sistema es parecido al C3 -AP-, pero con menos réplicas -N=7-. Es cierto que se cumple la inecuación Al primar la disponibilidad, se descartan los casos 1 y 3 en favor del **caso 2**, que es donde mejor encajaría. Como factor diferenciador del sistema C3, al tener menos nodos se mejora la consistencia al simplificar la actualización de las réplicas. Esto se debe a que se cumple la inecuación ***W > N/2*** pero, al no cumplirse ***W + R > N***, la consistencia no prima frente a la disponibilidad.

C6: N=6, W=6, R=1

Como ***W > N/2*** y ***W + R > N***, se trata de un sistema CP de consistencia fuerte. Por ello, se descarta el caso 2, ya que se sacrifica la disponibilidad en favor de la consistencia.

Por lo tanto, sí encajaría con los casos 3 y 1; aunque, en este último, donde se les da la misma importancia a las operaciones de lectura que a las de escritura, este sistema estaría infradimensionado con ***R=1***. Por lo tanto, el **caso 3** sí encaja perfectamente ya que se les da más peso a las operaciones de escritura que de lectura.

Propiedad intelectual

Esta PEC ha sido realizada por Javier Gómez de Diego.

Todos los recursos utilizados y citados en la realización de esta PEC han sido provistos por la UOC.

Referencias

[1] M. E. R. González y J. C. i. Caralt, «B3_T5_3_BDD_Diseño».

[2] M. E. R. González, «Gestión de Transacciones,» UOC.

[3] M. E. R. González y J. C. i. Caralt, «B5_T6_BDD_ACID,» UOC.

[4] M. E. R. Gónzalez, J. C. i. Caralt y P. U. Bayers, «B2_T3_2_ModelosAgregacionCaracteristicas».