Distribución de datos y *Map Reduce*
PEC2

Ejercicio 1 (25%)

En una organización se almacenan y gestionan series temporales (datos indexados en orden temporal) del tipo `{timestamp, métrica, valor, origen, tags}`. Se generan miles de datos por minuto, al final del día hay millones de registros. El tipo de datos generado, es decir los atributos generados y su formato es uniforme y no se esperan cambios a corto plazo. Se supone que los datos son consultados de forma concurrente por multitud de usuarios. Los datos no requieren un almacenamiento indefinido, ya que se utilizan para estudiar el estado de los sistemas.

Se pide:
Estudiar la conveniencia de un modelo relacional o uno NoSQL y razonar qué modelo sería el más adecuado.

A partir de la arquitectura de distribución escogida:

1. Indicar las ventajas e inconvenientes que tiene la estrategia escogida respecto a otras por lo que respecta a la fragmentación (partición) de los datos.
2. Indicar las ventajas e inconvenientes que tiene la estrategia escogida respecto a otras por lo que respecta a la replicación de los datos.
3. El modelo transaccional más adecuado para los requisitos de la aplicación descrita.

Exponed la solución de forma argumentada en una página como máximo.

Solución:

Modelo de datos

La descripción del sistema indica que será necesario gestionar un elevado número de datos y habrá una elevada concurrencia, siendo necesario minimizar el tiempo que transcurre desde que se produce el dato hasta que está disponible para lectura.
Por lo tanto, un modelo relacional no será la solución más adecuada para procesar tal cantidad de datos o con la frecuencia en la que se producen los datos. Tampoco sería adecuado un modelo en grafo dado que no existen muchas relaciones entre los datos, ni hay que navegar por ellas. La mejor opción sería utilizar un modelo orientado hacia

agregados (clave-valor, orientada a documentos u orientada a columnas) que ofrece alta disponibilidad, facilidad para la fragmentación horizontal de los datos y soporte de elevados niveles de concurrencia.

Respecto a la fragmentación (particionamiento) de los datos

Con el objetivo de facilitar el tratamiento de los datos, éstos se distribuirán a través de la red en tantos agregados como sean necesarios para llevar a cabo su procesamiento. Concretamente será necesario agrupar los datos en rangos de tiempo para hacer correlaciones entre eventos. Por tanto cada fragmento (shard) tendría una ventana de duración predefinida. Los datos de series temporales por naturaleza podrían ser descartados o almacenarse de forma resumida con el paso del tiempo (en función de la política de retención que se defina).

Respecto a la replicación de los datos

Con el objetivo de asegurar que se siguen procesando los datos incluso cuando hay caídas en los nodos del sistema, los datos se replicarán al menos una vez. Con una estrategia de replicación de tipo maestro-esclavo (master-slave) asíncrona se podría conseguir un alto rendimiento que permitiría responder a un elevado número de accesos a los datos. En caso de esperarse accesos de consulta desde distintos puntos geográficos determinados, podría incrementarse la fragmentación/replicación para considerar distintos servidores geográficamente distribuidos que acerquen los datos allí donde se necesiten.

Respecto al modelo transaccional

Debido a la naturaleza de los datos, un sistema transaccional de tipo ACID no es lo más adecuado. Sería más útil un modelo BASE donde se prime la disponibilidad de los datos a la consistencia debido a la propia naturaleza de los mismos. Sería asumible un pérdida temporal de la consistencia.

Ejercicio 2 (25%)

A partir de la lectura del artículo 'Consistency Models of NoSQL Databases' y de los apuntes indica si te parecen ciertas o falsas las siguientes afirmaciones.

Para cada una de las afirmaciones indica si es cierta o falsa, justificando la respuesta mediante lo que has leído en el artículo. En cada justificación deberá **indicar el párrafo del artículo** en la que se sustenta tu argumentación.

No serán válidas las respuestas que no se justifiquen.
Se valorará la concisión (una página y media para las 5 afirmaciones como máximo).

Afirmación 1

Según el teorema CAP se puede afirmar que si una base de datos es CA implica que los datos son consistentes entre todos los nodos (mientras los nodos estén en línea) y que se puede leer/escribir de forma consistente en cualquier nodo puesto que los datos serán los mismos.

Afirmación 2

Las bases de datos orientadas a documentos sólo admiten replicación, resultando imposible las técnicas de distribución como sharding.

Afirmación 3

La consistencia final en el tiempo establece que todas las réplicas llegarán a ser gradualmente consistentes si no hay actualizaciones.

Afirmación 4

La consistencia fuerte garantiza que una operación de lectura para un dato obtendrá el valor de la última escritura, aunque alguna de estas réplicas pueda tener valores inconsistentes.

Solución:

Afirmación 1

Cierta. En este contexto todos los nodos mantendrían los datos actualizados en todo momento, aunque el sistema dejaría de estar disponible frente a posibles caídas de red. Este tipo de base de datos distribuidos raramente se da pues mantener la consistencia es costoso. B3_T7_BDD_BASE, página 21: “las bases de datos distribuidas relacionales son sistemas CA.” (...) “Recordamos que el sistema gestor de la base de datos (al menos desde un punto de vista teórico) debe proveer consistencia estricta (strict consistency).” (...) “también garantiza que las modificaciones efectuadas sobre datos replicados se aplicarán en todas las réplicas antes de dar por finalizada la transacción.”

Afirmación 2

La afirmación es falsa. Como se afirma en la transcripción de los apuntes (Bases de datos distribuidas): “el caso de bases de datos orientadas a documentos, el sharding se puede efectuar, bien aplicando técnicas de hash o bien a partir del valor que toman ciertos atributos”. Documento B3_T5_3_BDD_Disenyo.pdf, página 14

Afirmación 3

La afirmación es cierta. El artículo indica en la página 4, segundo párrafo (en medio de las dos figuras): "The Eventual Consistency dimensión states that all réplicas will gradually become consistente if no update operation occurs".

Afirmación 4

Cierta. Es la definición de consistencia fuerte. Documento B3_T7_BDD_BASE.pdf, página 16: “Para preservar la consistencia se deben dar dos condiciones. La primera, tal y como muestra la inecuación [1], es que es necesario escribir de forma atómica al menos la mitad más una de las réplicas que de unos mismos datos existen. Esto evita que se puedan perder cambios y asegura que al menos existe una réplica que contiene el valor correcto (es decir, el más recientemente confirmado).” (…) “La segunda condición (véase la inecuación [2]) garantiza que, a efectos de operaciones de lectura, siempre se recuperará, al menos, una réplica que contendrá el valor correcto. Esto es así porque entre los conjuntos de réplicas accedidos por escritura y lectura siempre existe intersección.”

Ejercicio 3 (30%)

Ejecuta el archivo `IntroMapReduce.ipynb` que se entrega junto al enunciado de la PEC siguiendo las instrucciones descritas en el anexo I. A continuación, resuelve los dos supuestos que se proponen. No es necesario que codifiques en Python la solución, basta que expliques tu solución utilizando pseudocódigo, explicaciones textuales así como los datos proporcionados para los resultados.

3.1 Suponer los datos de una red social que consisten en un conjunto de pares del tipo (personaA, personaB) que representan una relación “sigue a” (‘following’) de forma que la personaA sigue a la personaB. Dado el siguiente conjunto de datos,describe el algoritmo MapReduce que calcula el número de seguidores que tiene cada persona.

```
red_social = [('Alicia', 'Benito'), ('Benito', 'Alicia'),
              ('Carlos', 'Benito'), ('Benito', 'Carlos'),
              ('Daniela', 'Enrique'), ('Enrique', 'Francisco'),
              ('Francisco', 'Enrique'), ('Daniela', 'Benito')]
```

3.2 La relación “sigue a” no es simétrica, ya que una persona no tiene porqué seguir a sus seguidores. No obstante, a veces ocurre que una persona sigue a la persona que lo sigue. Es decir, si PersonaA sigue a PersonaB, entonces PersonaB sigue a PersonaA. Con este ejercicio queremos identificar los pares (PersonaA, PersonaB) que no tienen una relación (PersonaB, PersonaA) definida. Con los mismos datos anteriores, describe el algoritmo MapReduce que permite obtener la lista de las relaciones que cumplen dicha condición.

Solución:

3.1 El proceso map

```
procedure Map ( lista )
  for par in lista do:
    person = par[1] // par[0] --> par[1]
    emit ( person, 1 )
```

El procedimiento emitirá:

```
{ 'Benito' : 1 , 'Alicia' : 1,  'Benito': 1 , 'Carlos': 1,
  'Enrique': 1 , 'Francisco': 1, 'Enrique': 1, 'Benito': 1}
```

En una fase intermedia 'shuffle' se agruparían los pares por clave, quedando:

```
{ 'Benito': [ 1,1,1 ] , 'Alicia': [ 1 ], 'Carlos': [ 1 ] ,
  'Enrique': [ 1, 1 ], 'Francisco': [ 1 ] }
```

En la fase de reduce:

```
procedure Reduce (persona , valores ):
  emit( (person, len(valores))  // nº de elementos / sum
```

El resultado final sería:

```
 [ ('Benito', 3) , ('Enrique', 2), ('Alicia', 1), ('Carlos', 1),
  ('Francisco': 1 ) ]
```

3.2 El proceso map

```
procedure Map ( lista )
  for par in lista do:
    personaA = par[0]
    personaB = par[1]
    emit ( personaA+personaB, par)
    emit ( personaB+personaA, (par[1], par[0]))
```

El procedimiento emitirá:

```
{'AliciaBenito': ('Alicia','Benito'),
 'BenitoAlicia': ('Benito','Alicia'),
 'BenitoAlicia': ('Benito','Alicia'),
```

```
 'AliciaBenito': (‘Alicia’,’Benito’),
 'CarlosBenito': ('Carlos', 'Benito'),
 'BenitoCarlos': ('Benito', 'Carlos'),
 'BenitoCarlos': ('Benito', 'Carlos'),
 'CarlosBenito': ('Carlos', 'Benito'),
 ‘DanielaEnrique': ('Daniela', 'Enrique'),
 ‘EnriqueDaniela': ('Enrique', 'Daniela'),
 'EnriqueFrancisco': ('Enrique', 'Francisco'),
 'FranciscoEnrique': ('Francisco', 'Enrique'),
 'FranciscoEnrique': ('Francisco', 'Enrique'),
 'EnriqueFrancisco': ('Enrique', 'Francisco'),
 'DanielaBenito': ( 'Daniela', 'Benito')
 'BenitoDaniela': ('Benito', 'Daniela') }
```

Agrupando los datos quedaría:

```
{ 'AliciaBenito': [ (‘Alicia’,’Benito’),(‘Alicia’,’Benito’) ],
  ‘BenitoAlicia’: [ (‘Benito’,’Alicia’), (‘Benito’,’Alicia’) ],
  'CarlosBenito': [ ('Carlos', 'Benito'),  ('Carlos', 'Benito') ],
  'BenitoCarlos': [ ('Benito', 'Carlos'),('Benito', 'Carlos') ],
  ‘DanielaEnrique': [('Daniela', 'Enrique') ],
  ‘EnriqueDaniela': [('Enrique', 'Daniela') ],
  'EnriqueFrancisco': [  ('Enrique', 'Francisco'),('Enrique', 'Francisco')],
  'FranciscoEnrique': [('Francisco', 'Enrique'), ('Francisco', 'Enrique')],
  'DanielaBenito': [('Daniela', 'Benito')],
  'BenitoDaniela': [('Benito', 'Daniela') ]
}
```

La parte reduce es un poco más complicada pues hay que contar el número veces que la persona aparece en la lista y si es inferior a 2, entonces no hay relación simétrica:

```
procedure Reduce (persona , lista):
  _count = len ( lista )
  if _count == 1 :
    emit ( (lista[0][0], lista[0][1]) )
```

Para obtener finalmente los que no tienen relaciones simétricas:

```
[ ('Benito', ‘Daniela’), (Daniela, Benito),
  (‘Enrique’, ‘Daniela) (Daniela, Enrique) ]
```

Ejercicio 4 (20%)

Suponiendo que debemos configurar el sistema de replicación de una base de datos para que cumpla la condición de consistencia fuerte y se nos plantean distintas alternativas. Sabemos que se reciben muchas solicitudes de lectura y relativamente pocas de escritura. ¿Cuál de las siguientes alternativas de quórum elegirías y por qué?

1. N=7, R=3, W=1
2. N=7, R=1, W=7
3. N=7, R=5, W=4
4. N=7, R=4, W=5

Solución:

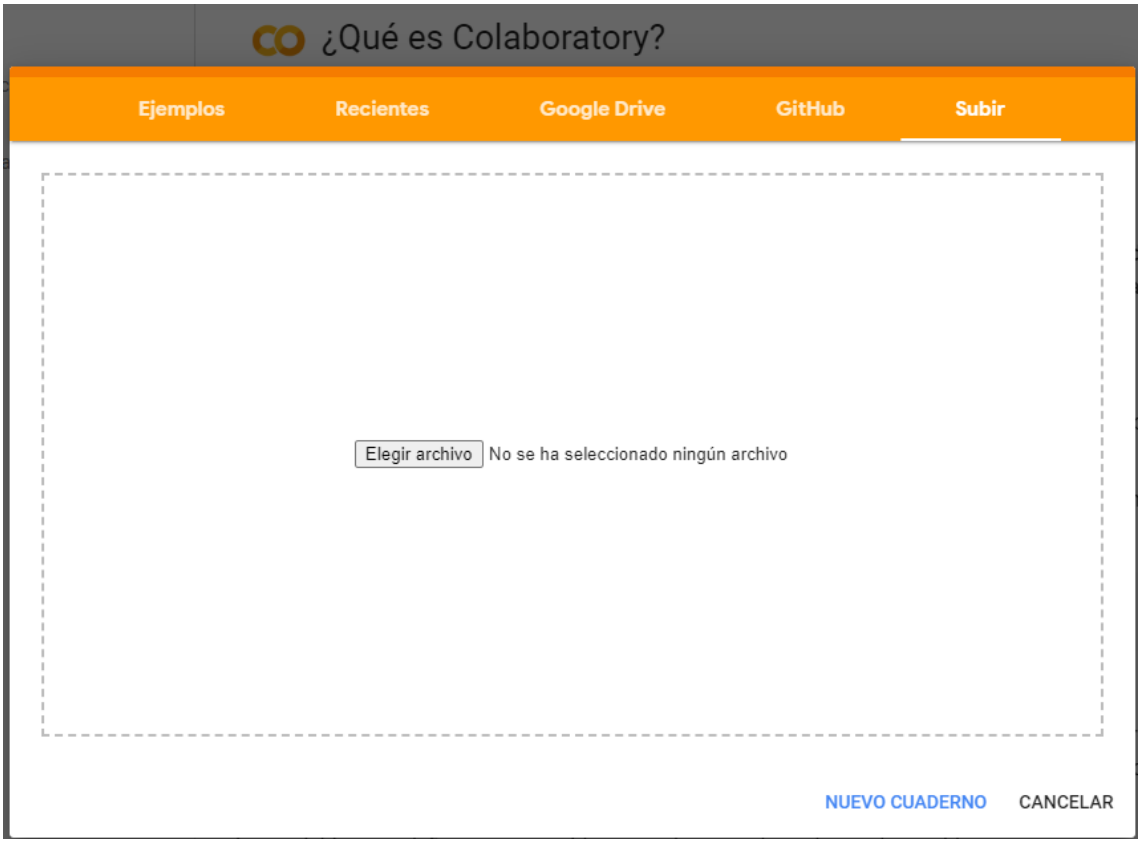
1. Este sistema ofrecería consistencia final en el tiempo, por tanto no satisfaría los requerimientos propuestos.
2. Esta podría ser una opción válida, ya que cumple con una consistencia estricta o fuerte (N < W + R [7 <1 + 7] y W > N/2 [7 > 7/2]). De hecho, es equivalente a un modelo de replicación de propagación de cambios de manera síncrona. No obstante, el sistema tendría poca disponibilidad ya que al caer un nodo, se rechazarían las operaciones de escritura. Por tanto, cumple con la consistencia fuerte pero es poco aconsejable si hay mejores opciones.
3. Este sistema ofrece consistencia fuerte (N < W + R [7 < 5 + 4] y W > N/2 [4 > 7/2]). Además, este sistema ofrece una disponibilidad más alta que el sistema anterior, ya que permitiría operar aún cuando 2 réplicas estuvieran inaccesibles (o caídas). Por lo tanto, podría ser una opción válida.
4. Este sistema ofrece consistencia fuerte (N < W + R [7 < 4 + 5] y W > N/2 [5 > 7/2]). Además, este sistema ofrece una disponibilidad similar al sistema anterior, pero como hay más lecturas que escrituras es más eficiente que el anterior en el caso de interés, ya que debe considerar menos nodos en cada lectura. Por lo tanto, se escogería esta opción como válida.

Anexo I

Descarga en tu equipo el notebook `IntroMapReduce.ipynb`. El notebook está pensado para que lo ejecutes en Google Colab (https://colab.research.google.com/). **Google Colab** es un servicio en la nube que permite ejecutar Jupyter Notebooks accediendo con un navegador web. Tiene además las siguientes ventajas:

- Posibilidad de ejecución mediante GPUs
- Basado en Jupyter Notebook pudiendo crear y ejecutar libros en Python 2 o 3
- Tiene preinstaladas las librerías comunes usadas en ciencia de datos y la posibilidad de instalar otras.
- Enlaza con cuentas de Google Drive y desde github

Primero hay que entrar en sesión (login) con una cuenta de Google (la de la uoc debería funcionar). Ahora ya se puede subir el notebook de la PEC a la plataforma:



La ejecución del libro es exactamente igual que en cualquier Jupyter Notebook. Hay que pulsar Shift + Enter para que el código (python) se ejecute. Simplemente vaya ejecutando cada celda y vea el resultado de la ejecución del código.

Criterios de valoración

Los apartados 1 y 2 tienen un peso del 25% cada uno, y los apartados 3 y 4 tienen un peso del 30% y el 20% respectivamente. Se valorará, para cada apartado, la validez de la solución y la claridad de la argumentación. Cualquier solución no justificada se considerará incompleta.

Formato y fecha de entrega

Tenéis que enviar la PEC al buzón de Entrega y registro de EC disponible en el aula (apartado Evaluación). El formato del archivo que contiene vuestra solución puede ser .pdf,. odt, .doc y .docx. Para otras opciones, por favor, contactar previamente con vuestro profesor colaborador. El nombre del fichero debe contener el código de la asignatura, vuestro apellido y vuestro nombre, así como el número de actividad (PEC2). Por ejemplo nombreapellido1_nosql_pec1.docx. La fecha límite para entregar la PEC2 es el **17 de noviembre.**

Propiedad intelectual

Al presentar una práctica o PEC que haga uso de recursos ajenos, se tiene que presentar junto con ella un documento en que se detallen todos ellos, especificando el nombre de cada recurso, su autor, el lugar donde se obtuvo y su estatus legal: si la obra está protegida por el copyright o se acoge a alguna otra licencia de uso (Creative Commons, licencia GNU, GPL etc.). El estudiante tendrá que asegurarse que la licencia que sea no impide específicamente su uso en el marco de la práctica o PEC. En caso de no encontrar la información correspondiente tendrá que asumir que la obra está protegida por el copyright.

Será necesario, además, adjuntar los ficheros originales cuando las obras utilizadas sean digitales, y su código fuente, si así corresponde.