

**Máster universitario de Ciencia de Datos**

**Prueba de Evaluación Continua – PEC2**

**Arquitecturas de bases de datos no tradicionales – Distribución de datos y *Map Reduce*.**

Autor:

Mario Ubierna San Mamés

|  |
| --- |
| Índice de Contenido |

[Índice de Contenido 2](#_Toc87169035)

[1. Enunciado 3](#_Toc87169036)

[1.1. Ejercicio 1 3](#_Toc87169037)

[2. Bibliografía 4](#_Toc87169038)

|  |
| --- |
| Enunciado |

## Ejercicio 1

*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.*

### Modelo relacional o modelo NoSQL.

Vemos que el problema busca almacenar millones de registros diarios y hay un gran número de usuarios/operaciones de lectura de forma concurrente, a priori son características de un modelo NoSQL, pero la estructura de los datos está definida y no va a variar a corto plazo, característica más adecuada para el modelo relacional. En resumen, considero ambos modelos pueden servir pero creo que lo mejor es un modelo relacional, ya que éste también puede almacenar millones de registros y permitir la concurrencia de usuarios/operaciones de forma óptima manteniendo una estructura fija de los datos.

En cuanto a la arquitectura de distribución elegida es *peer-to-peer*, ya que con el enunciado no queda claro cuántos clientes/servidores vamos a tener y dónde se encuentran éstos, por lo que no podemos definir un modelo cliente/servidor o tres capas. En *peer-to-peer* todos los nodos son cliente/servidor por lo que nos evitamos el problema mencionado antes, también se generan millones de registros diarios y hay muchos usuarios, por lo que este tipo de arquitectura se comporta mejor, ya que hay muchísimos nodos en la red y las operaciones se ejecutan a lo largo de toda la red no en un servidor o conjuntos de servidores limitados.

### Ventajas/inconvenientes respecto a la fragmentación

Puesto que no hay un gran número de atributos (5) considero que lo mejor es la fragmentación horizontal, teniendo como ventaja que las consultas de los propios usuarios nos indiquen qué fragmentos se necesitan crear, también tiene como ventaja respecto a la fragmentación vertical que no necesita crear una relación entre registros de un mismo fragmento, mejorando así inserción de datos, ni hacer un estudio de la afinidad entre atributos para el diseño de un fragmento. El principal inconveniente existente es que hay que definir muy bien el fragmento en sí, ya que si se realiza un consulta y dicha fragmentación no contiene toda la información necesaria hay que seguir buscando por la red para encontrar la información, esto supone coste computacional y temporal.

### Ventajas/inconvenientes respecto a la replicación

La estrategia elegida para la replicación de datos es replicación P2P con quórums, esta estrategia se comporta mejor que *master-slave* ya que no genera cuellos de botella en la copia maestra (*master-slave* es ideal cuando hay más lecturas que escrituras, pero en nuestro caso es al contrario, es verdad que la escritura a priori se va a hacer solamente una vez ya que es una serie temporal, pero según el enunciado se busca “gestionar” las series temporales, por lo que supongo que se pueden hacer *updates* sobre los registros). Dentro de la replicación P2P hay diferentes políticas, se ha elegido quórums porque tiene como ventaja un mejor rendimiento que la política síncrona (no tiene que escribir de forma atómica todo) y una mejor consistencia que la política asíncrona. Sin embargo, la principal desventaja de esta estrategia es que tenemos que definir el número de réplicas escritas de forma atómica y el número de réplicas recuperadas para la lectura, identificar los parámetros óptimos para esta estrategia no es fácil.

### Modelo transaccional adecuado

Puesto que en la primera parte del ejercicio se ha definido que el modelo que mejor se ajusta es un modelo relacional, entonces el modelo transaccional adecuado sería el modelo ACID. Éste nos garantiza la consistencia de los datos, que a partir de un sistema distribuido también obtenemos la alta disponibilidad de los mismos. El modelo ACID incluye protocolos para la gestión de las réplicas y de la concurrencia que hemos visto que pueden existir según los requisitos de la aplicación descrita.

|  |
| --- |
| Bibliografía |