NoSQL - Map-Reduce - Concepto

Map-Reduce

MapReduce es un marco de trabajo para el procesamiento en paralelo de grandes volúmenes de datos en varios equipos (nodos).

Funcionamiento

- La operación **map** tiene un nodo principal, que divide una operación en subpartes y distribuye cada operación a otro nodo para su procesamiento,
- y **reduce** es el proceso donde el nodo maestro junta los resultados de los otros nodos y las combina en respuesta al problema original.

NoSQL - Map-Reduce - Idea Programación Funcional

Idea de Programación Funcional

La idea proviene de programación funcional

Ejemplo de Programación Funcional

- **map** aplica una función a cada elemento de una lista. Ejemplo: Si la función a aplicar es *duplicate* y la lista es $[1, 2, 3, 4]$, al aplicar dicha función utilizando **map**, devuelve $[2, 4, 6, 8]$ (la lista original no se altera)
- Notar que es altamente paralelizable.
- **reduce** (o fold) es una función de agrupamiento o acumulación de los elementos de la lista. Ejemplo: Si la función es *suma* y la lista es la generada por el **map** previo ([2,4,6,8]), entonces el **reduce** devuelve 20.

NoSQL - Map-Reduce - Idea BD

Idea en BD

Misma idea, pero aplicado a set de datos

Ejemplo en BD

En una BD orientada a clave/valor:

- **map** aplica una función a cada par (clave,valor), generando una nueva colección.
- **reduce** aplica una función de agregación (ej: suma) a la colección generada por **map**, para devolver el resultado final.

NoSQL - Map-Reduce - Ejemplo

Ejemplo

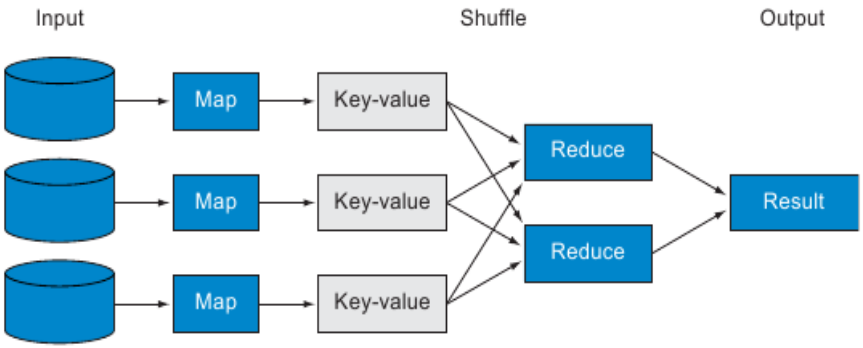

Figure 6.10 **The basics of how the map and reduce functions work together to gain linear scalability over big data transforms. The map operation takes input data and creates a uniform set of key-value pairs. In the shuffle phase, which is done automatically by the MapReduce framework, key-value pairs are automatically distributed to the correct reduce node based on the value of the key. The reduce operation takes the key-value pairs and returns consolidated values for each key. It's the job of the MapReduce framework to get the right keys to the right reduce nodes.**

Figura de McCreary/Kelly-*Making Sense of NoSQL*, Manning, 2014

1. **map** recupera los datos de la BD y los transforma en una colección de operaciones que pueden ser ejecutados independientemente en distintos procesadores.
2. La salida de los **map**, son pares (clave,valor).
3. Como siguiente fase, **reduce**, utiliza los pares como entrada, ejecuta la operación asignada y retorna un resultado.

NoSQL - Map-Reduce - Problemas

Problema I

¿Qué pasa si los datos de origen se encuentran en tres o más nodos? ¿Se mueven los datos entre nodos?
Si se quiere ser eficiente, la respuesta es NO.
Entonces, se debe tener en cuenta en qué nodo debe ejecutarse la función **map/reduce**.

NoSQL - Map-Reduce - Problemas

Problema I

¿Qué pasa si los datos de origen se encuentran en tres o más nodos? ¿Se mueven los datos entre nodos?
Si se quiere ser eficiente, la respuesta es NO.
Entonces, se debe tener en cuenta en qué nodo debe ejecutarse la función **map/reduce**.

Problema II

¿Qué pasa si en medio de la operación falla alguno de los **map** / **reduce**?
¿Es necesario reiniciar todo el trabajo completo o se puede asignar sólo la parte que falló a otro nodo?

NoSQL - Sharding

Problema

A medida que la cantidad de datos aumenta, puede llegar un momento en que se alcanza la capacidad máxima del sistema.
Surge la necesidad de particionar los datos.

NoSQL - Sharding

Problema

A medida que la cantidad de datos aumenta, puede llegar un momento en que se alcanza la capacidad máxima del sistema.
Surge la necesidad de particionar los datos.

Sharding

Fragmentar la BD en fragmentos denominados **shards** y distribuirlos a través de los servidores disponibles. Conceptualmente, los **shards** comparten esquemas y colectivamente representan el total del dataset.

NoSQL - Sharding

Problema

A medida que la cantidad de datos aumenta, puede llegar un momento en que se alcanza la capacidad máxima del sistema.
Surge la necesidad de particionar los datos.

Sharding

Fragmentar la BD en fragmentos denominados **shards** y distribuirlos a través de los servidores disponibles. Conceptualmente, los **shards** comparten esquemas y colectivamente representan el total del dataset.

Funcionamiento

- En el pasado: Dar de baja el sistema. Actualmente: Se puede realizar con el sistema en funcionamiento.
- Algunos sistemas permiten realizarlo de manera automática y otros de manera manual.

NoSQL - Sharding - Visualmente

Gráficamente

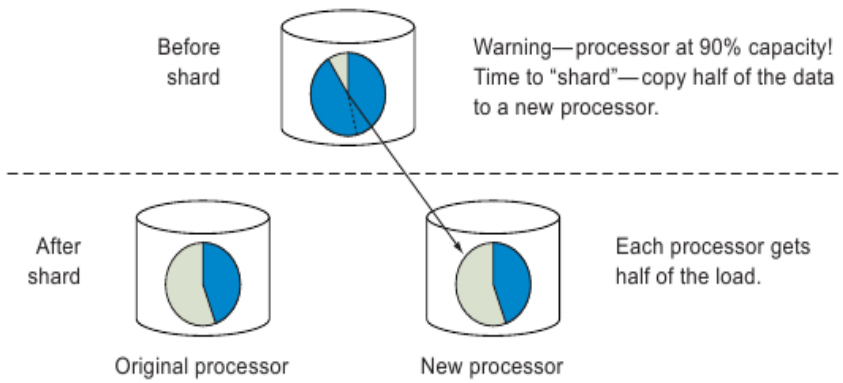

Figure 2.9 Sharding is performed when a single processor can't handle the throughput requirements of a system. When this happens you'll want to move the data onto two systems that each take half the work. Many NoSQL systems have automatic sharding built in so that you only need to add a new server to a pool of working nodes and the database management system automatically moves data to the new node. Most RDBMSs don't support automatic sharding.

Figura de McCreary/Kelly-*Making Sense of NoSQL*, Manning, 2014

NoSQL - Sharding - Ejemplo

Ejemplo

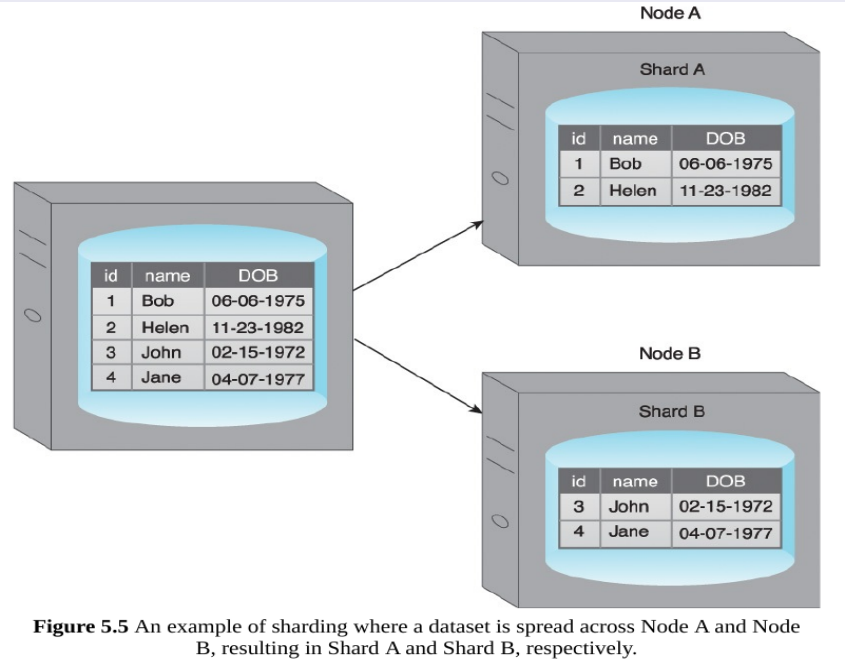

Figure 5.5 An example of sharding where a dataset is spread across Node A and Node B, resulting in Shard A and Shard B, respectively.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

NoSQL - Sharding - Ejemplo (Cont.)

En la práctica

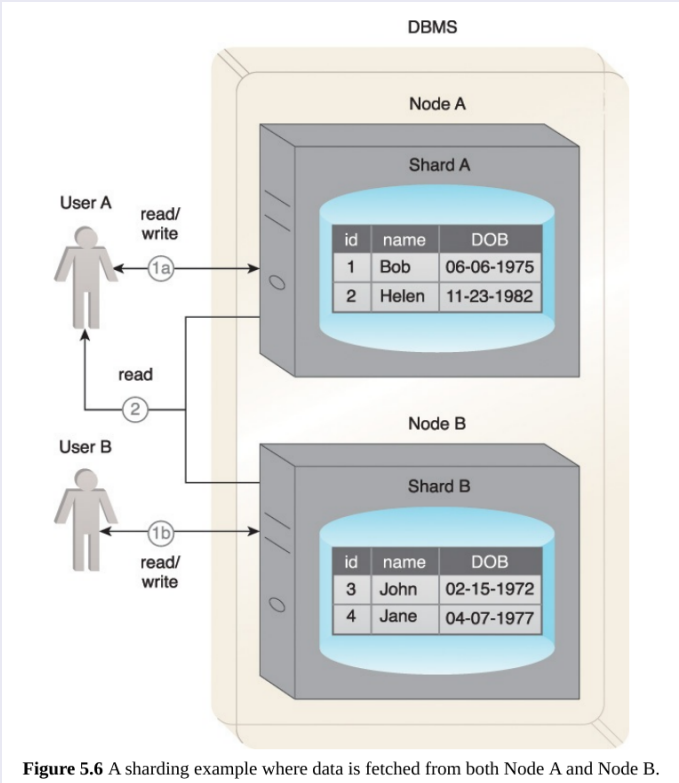

Figure 5.6 A sharding example where data is fetched from both Node A and Node B.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

NoSQL - Sharding - Ventajas/Desventajas

Ejemplos BD de cuentas de usuarios

Posibles criterios para fragmentar:

- Nombres de usuarios: A-N en un servidor y O-Z en otro
- Origen: división Continental
- Otra: Al azar

NoSQL - Sharding - Ventajas/Desventajas

Ejemplos BD de cuentas de usuarios

Posibles criterios para fragmentar:

- Nombres de usuarios: A-N en un servidor y O-Z en otro
- Origen: división Continental
- Otra: Al azar

Problemas

- ¿Si el usuario cambia de nombre?

NoSQL - Sharding - Ventajas/Desventajas

Ejemplos BD de cuentas de usuarios

Posibles criterios para fragmentar:

- Nombres de usuarios: A-N en un servidor y O-Z en otro
- Origen: división Continental
- Otra: Al azar

Problemas

- ¿Si el usuario cambia de nombre?
- ¿Si usuario se muda? Usuarios cercanos, ¿tienden a tener relaciones más estrechas? ¿Qué sucede, entonces, con determinados horarios (noche vs. día)? ¿sobrecarga de determinados servidores?

NoSQL - Sharding - Ventajas/Desventajas

Ejemplos BD de cuentas de usuarios

Posibles criterios para fragmentar:

- Nombres de usuarios: A-N en un servidor y O-Z en otro
- Origen: división Continental
- Otra: Al azar

Problemas

- ¿Si el usuario cambia de nombre?
- ¿Si usuario se muda? Usuarios cercanos, ¿tienden a tener relaciones más estrechas? ¿Qué sucede, entonces, con determinados horarios (noche vs. día)? ¿sobrecarga de determinados servidores?

Ventajas/Desventajas

- (+) Tolerancia parcial ante caída de nodos (sólo se pierden los datos del nodo afectado)
- (-) Consultas que involucran varios nodos pueden afectar negativamente la performance

NoSQL - Réplicas

Problema

A medida que aumenta la cantidad de servidores, aumenta la probabilidad de falla.

NoSQL - Réplicas

Problema

A medida que aumenta la cantidad de servidores, aumenta la probabilidad de falla.

Replicación

Almacenamiento de múltiples copias de la BD, cada una de ellas conocidas como **réplicas**.

NoSQL - Réplicas

Problema

A medida que aumenta la cantidad de servidores, aumenta la probabilidad de falla.

Replicación

Almacenamiento de múltiples copias de la BD, cada una de ellas conocidas como **réplicas**.

Funcionamiento: Gráficamente

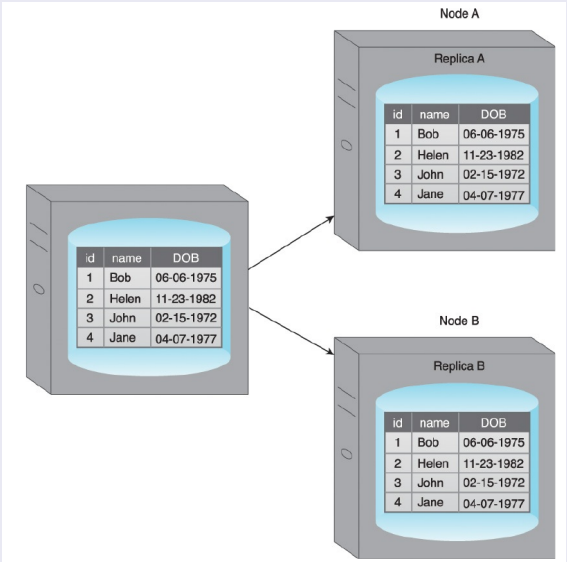

Figure 5.7 An example of replication where a dataset is replicated to Node A and Node B, resulting in Replica A and Replica B.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

NoSQL - Réplicas

Problema

A medida que aumenta la cantidad de servidores, aumenta la probabilidad de falla.

Replicación

Almacenamiento de múltiples copias de la BD, cada una de ellas conocidas como **réplicas**.

Métodos de implementación

- Master-Slave
- Peer-to-Peer

Funcionamiento: Gráficamente

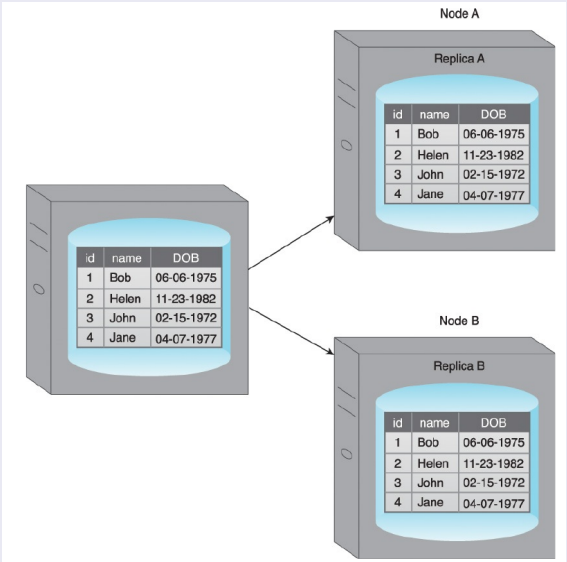

Figure 5.7 An example of replication where a dataset is replicated to Node A and Node B, resulting in Replica A and Replica B.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

NoSQL - Réplicas - Master-Slave

Funcionamiento: Gráficamente

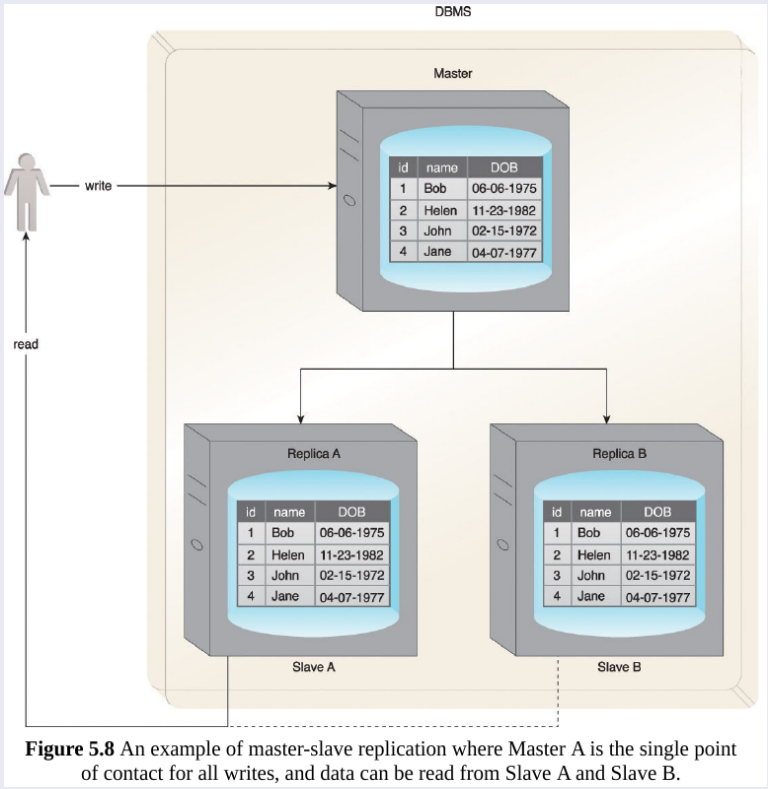

Figure 5.8 An example of master-slave replication where Master A is the single point of contact for all writes, and data can be read from Slave A and Slave B.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

Funcionamiento

- Escrituras (insert, delete, update) en el nodo Master.
- Lecturas sobre cualquier nodo Slave.

NoSQL - Réplicas - Master-Slave

Funcionamiento: Gráficamente

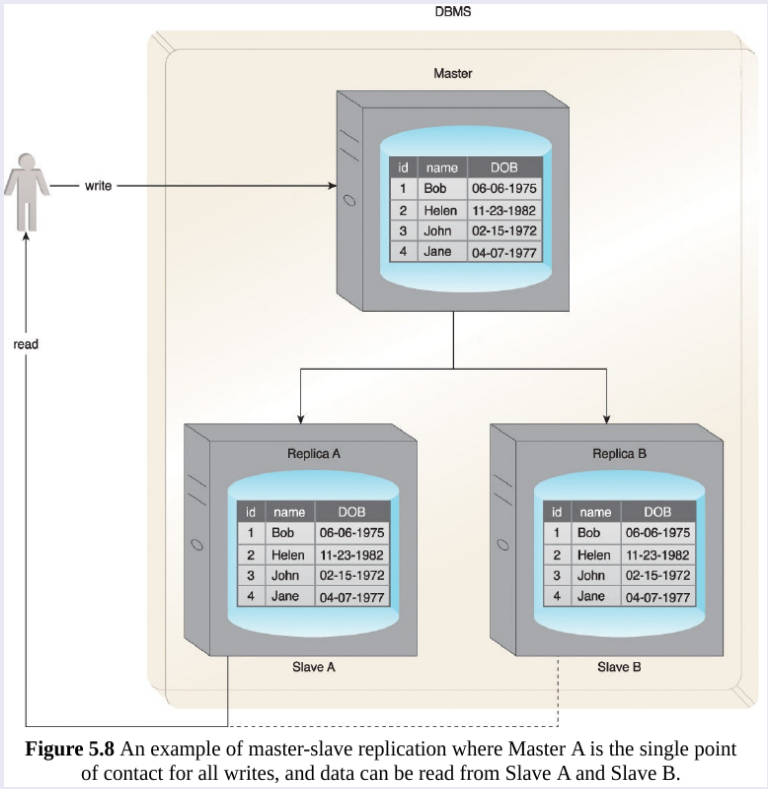

Figure 5.8 An example of master-slave replication where Master A is the single point of contact for all writes, and data can be read from Slave A and Slave B.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

Funcionamiento

- Escrituras (insert, delete, update) en el nodo Master.
- Lecturas sobre cualquier nodo Slave.

Ventajas/Desventajas

- (+) Ideal para escenarios de uso intensivo de lecturas
- (+) Si el nodo Master falla, se puede continuar las lecturas
- (+) Nodo Slave puede ser configurado como backup y tomar el rol de Master en caso de fallo

NoSQL - Réplicas - Master-Slave (Cont.)

Problema: Gráficamente

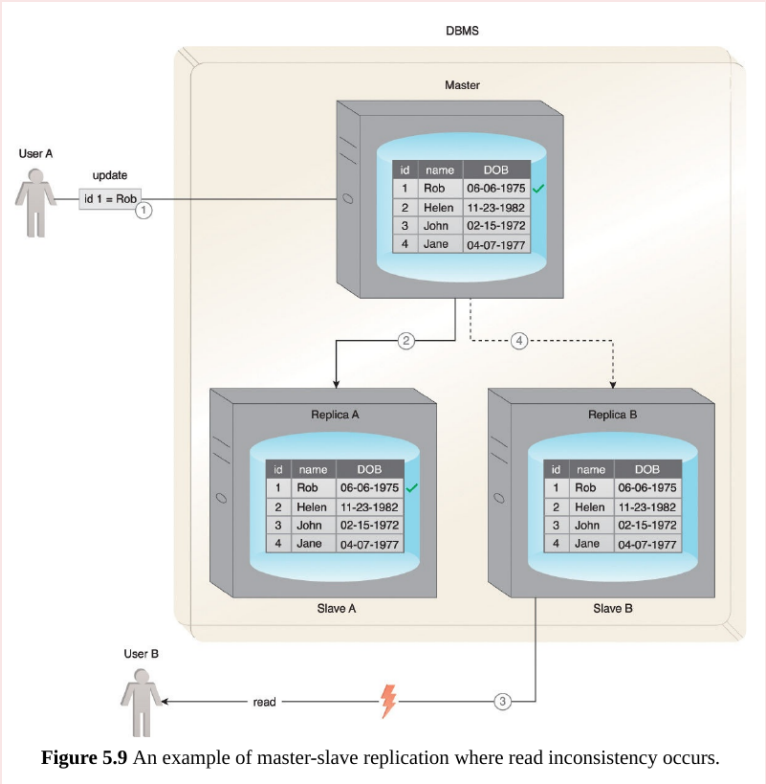

Figure 5.9 An example of master-slave replication where read inconsistency occurs.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

Problema

Lectura inconsistente

NoSQL - Réplicas - Master-Slave (Cont.)

Problema: Gráficamente

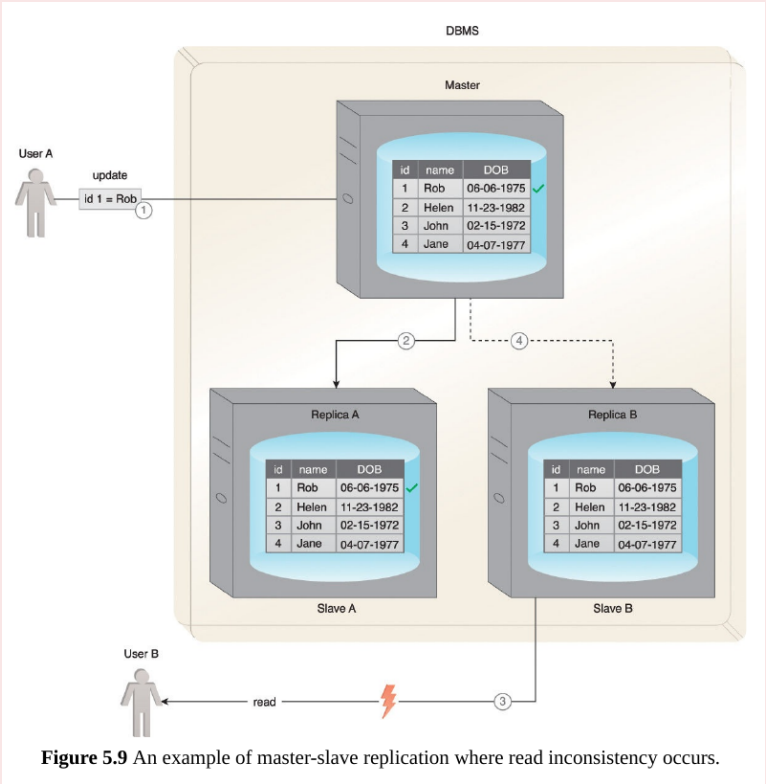

Figure 5.9 An example of master-slave replication where read inconsistency occurs.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

Problema

Lectura inconsistente

Posible Solución

Sistema de votación donde una lectura es consistente $\Leftrightarrow$ la mayoría de los nodos contienen la misma versión del registro.
Contra: Implementación requiere de un sistema de comunicación entre nodos Slaves rápido y confiable.

NoSQL - Réplicas - Peer-to-Peer

Funcionamiento: Gráficamente

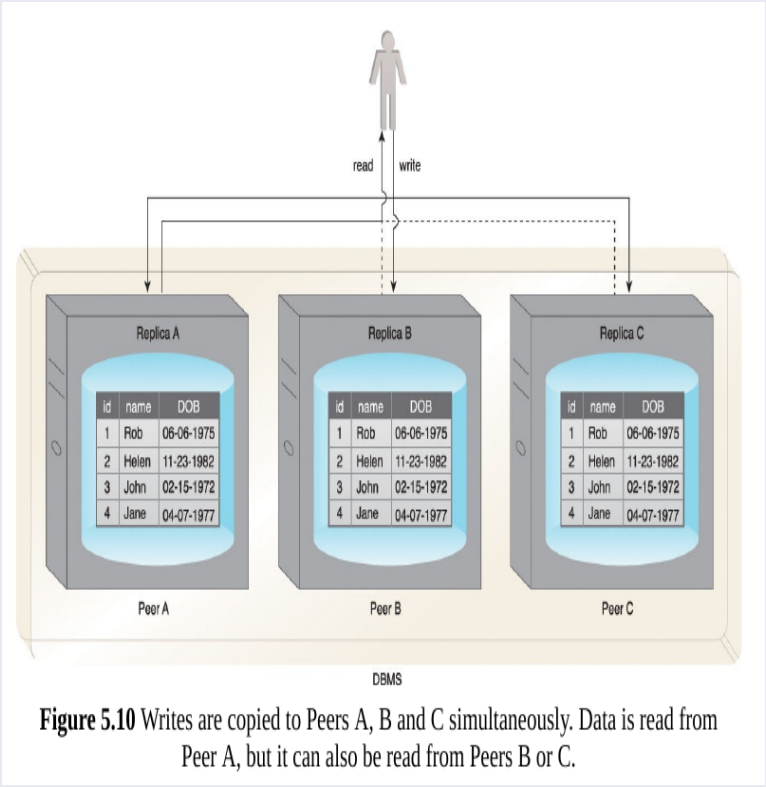

Figure 5.10 Writes are copied to Peers A, B and C simultaneously. Data is read from Peer A, but it can also be read from Peers B or C.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

Funcionamiento

Todos los nodos (denominados **peers**) possen el mismo nivel de jerarquía y son capaces de manejar tanto las lecturas como las escrituras.

NoSQL - Réplicas - Peer-to-Peer (Cont.)

Problemas: Gráficamente

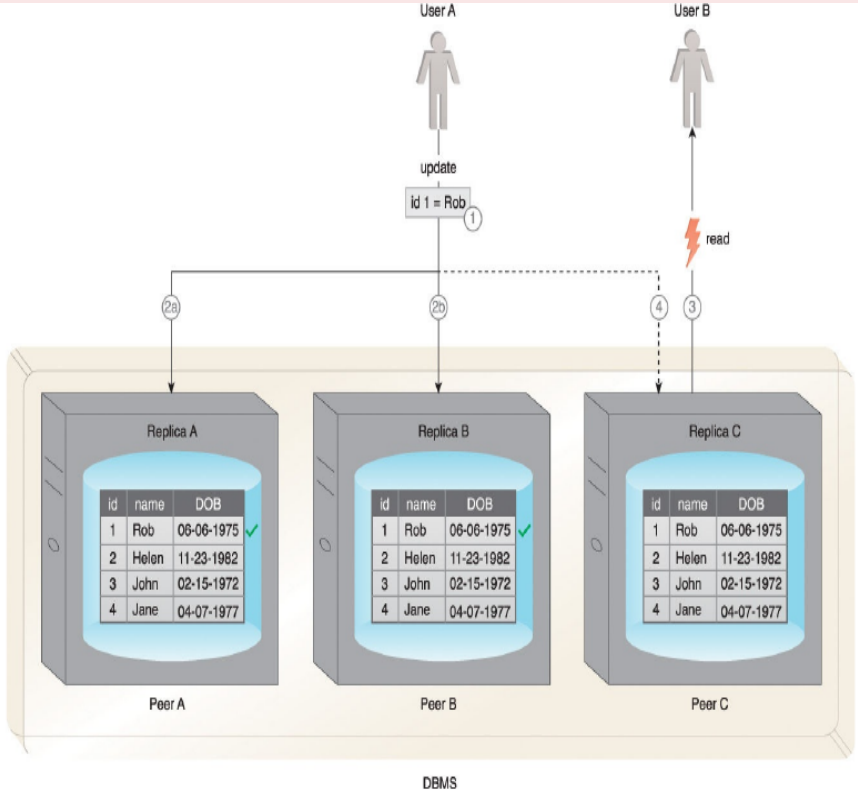

Figure 5.11 An example of peer-to-peer replication where an inconsistent read occurs.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

Problema

Lectura inconsistente

NoSQL - Réplicas - Peer-to-Peer (Cont.)

Problemas: Gráficamente

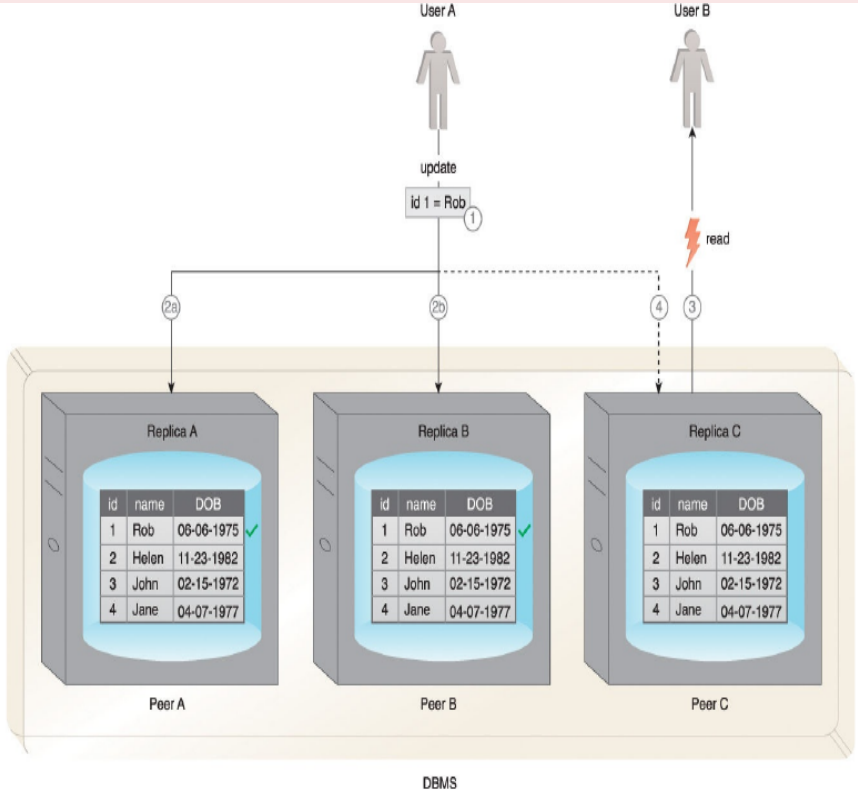

Figure 5.11 An example of peer-to-peer replication where an inconsistent read occurs.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

Problema

Lectura inconsistente

Estrategia de concurrencia Pesimista

- Estrategia proactiva.
- Utiliza **locking de registro**.
- Va en contra de la **disponibilidad**. El registro que está siendo actualizado permanece no disponible hasta que locks son eliminados.

NoSQL - Réplicas - Peer-to-Peer (Cont.)

Problemas: Gráficamente

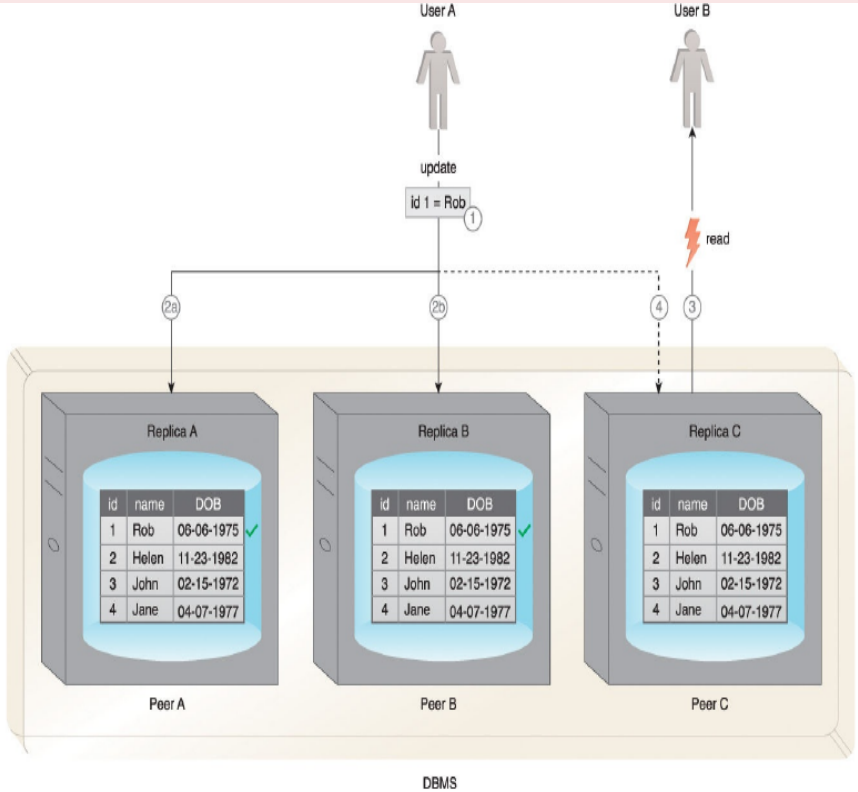

Figure 5.11 An example of peer-to-peer replication where an inconsistent read occurs.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

Problema

Lectura inconsistente

Estrategia de concurrencia Pesimista

- Estrategia proactiva.
- Utiliza **locking de registro**.
- Va en contra de la **disponibilidad**. El registro que está siendo actualizado permanece no disponible hasta que locks son eliminados.

Estrategia de concurrencia Optimista

- Estrategia reactiva.
- No utiliza **locking**.
- Permite inconsistencias sabiendo que eventualmente se llegará a un estado consistente luego de que TODAS las actualizaciones se propaguen.

NoSQL - Réplicas - Peer-to-Peer (Cont.)

Solución Optimista

- **Peers** pueden permanecer inconsistentes por un tiempo hasta alcanzar consistencia. Sin embargo, BD permanece disponible dado que no existen **locks**.
- **reads** pueden ser inconsistentes durante un tiempo. Mientras algunos **peers** completaron la actualización, otros están pendientes de hacerlo. Sin embargo, los **reads** eventualmente serán consistentes cuando la actualización alcance a TODOS los nodos.
- Para asegurar consistencia, se puede implementar un sistema de votación al igual que en el esquema Master-Slave

NoSQL - Sharding + Réplicas

Recordando . . .

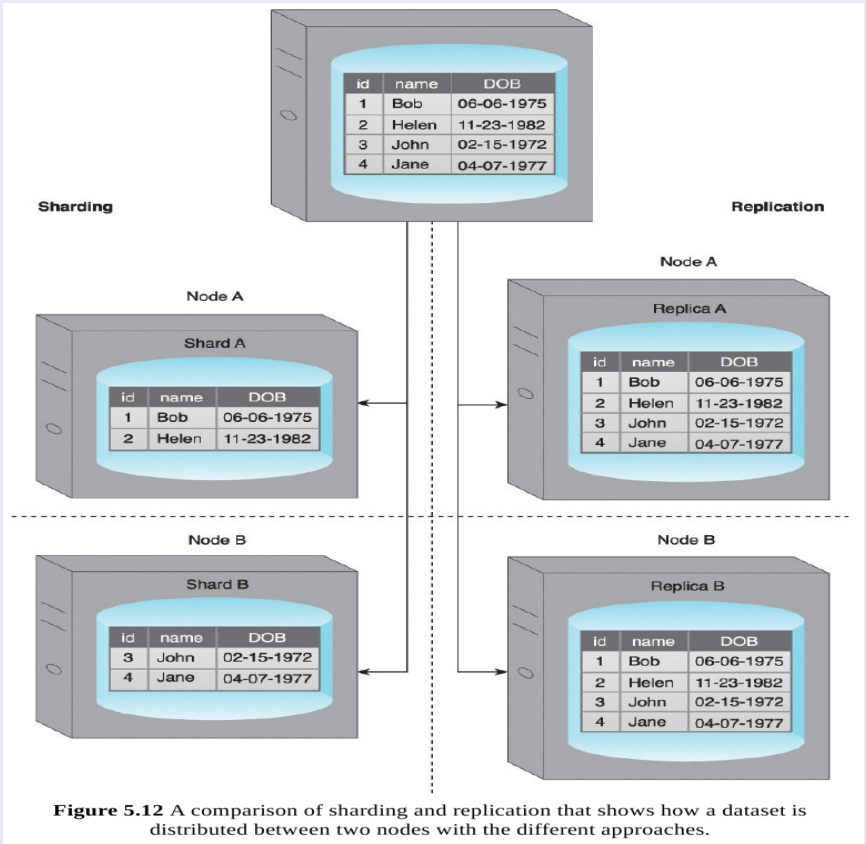

Figure 5.12 A comparison of sharding and replication that shows how a dataset is distributed between two nodes with the different approaches.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

NoSQL - Sharding + Réplicas

Recordando . . .

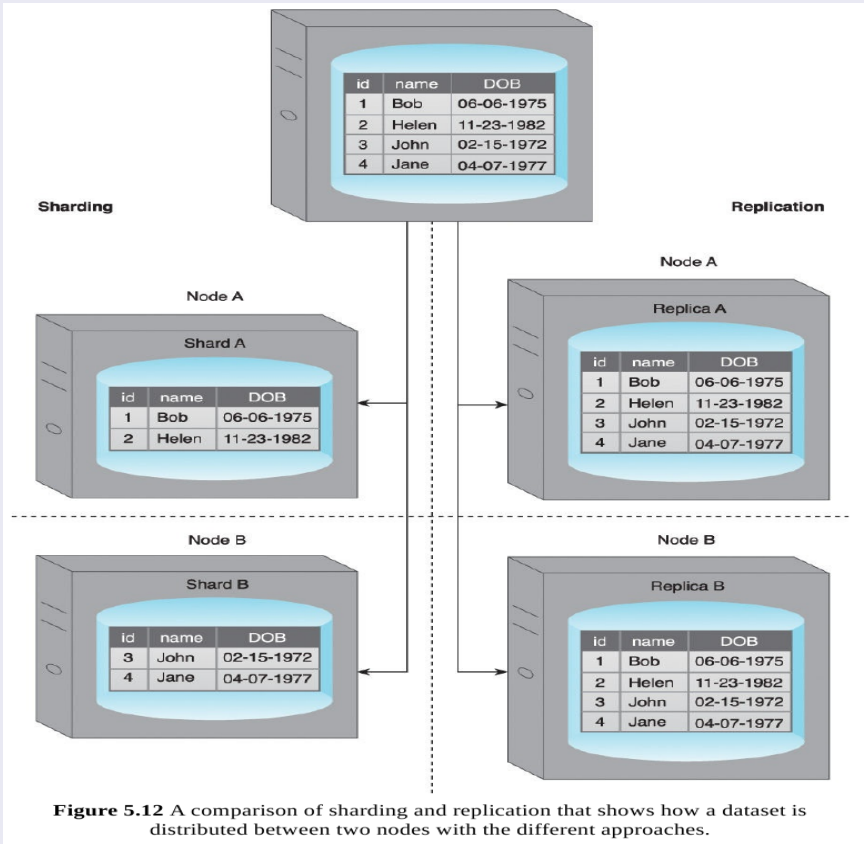

Figure 5.12 A comparison of sharding and replication that shows how a dataset is distributed between two nodes with the different approaches.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

Ventajas

- Mejorar la limitada tolerancia a fallos, ofrecida por sharding
- Aprovechar beneficios del incremento de disponibilidad y escalabilidad de las Réplicas.

NoSQL - Sharding + Réplicas

Recordando . . .

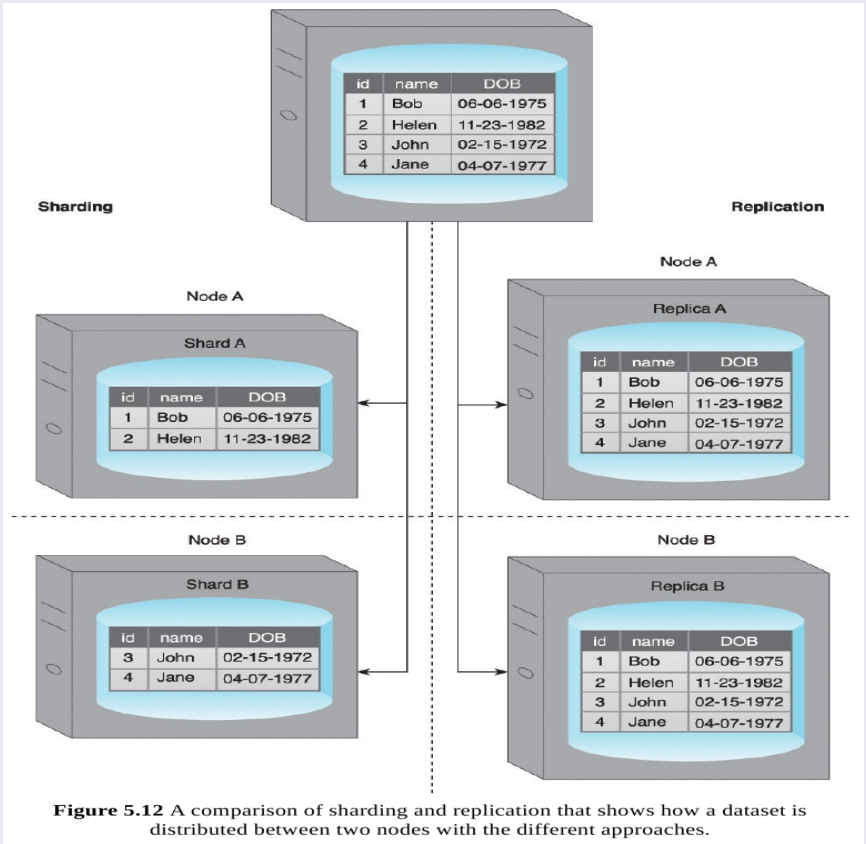

Figure 5.12 A comparison of sharding and replication that shows how a dataset is distributed between two nodes with the different approaches.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

Ventajas

- Mejorar la limitada tolerancia a fallos, ofrecida por sharding
- Aprovechar beneficios del incremento de disponibilidad y escalabilidad de las Réplicas.

Combinaciones

- Sharding + Replicación Master-Slave.
- Sharding + Replicación Peer-to-Peer.

NoSQL - Sharding + Réplicas - Master/Slave

Funcionamiento

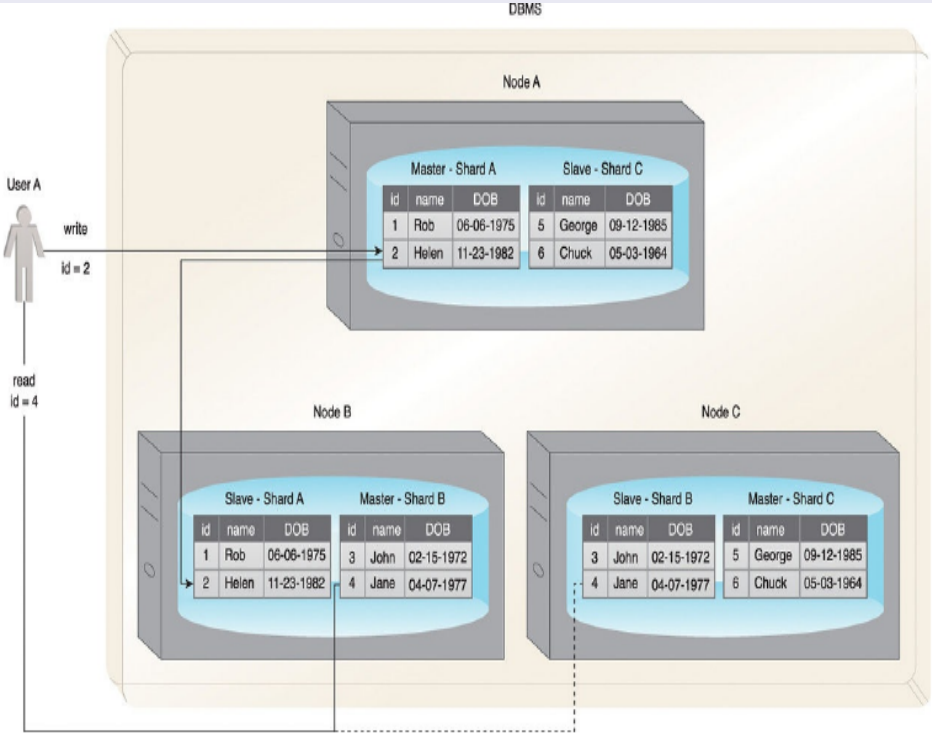

Figure 5.13 An example that shows the combination of sharding and master-slave replication.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

Funcionamiento

Esquema de Shard-Master y Shards-Slaves

NoSQL - Sharding + Réplicas - Master/Slave

Funcionamiento

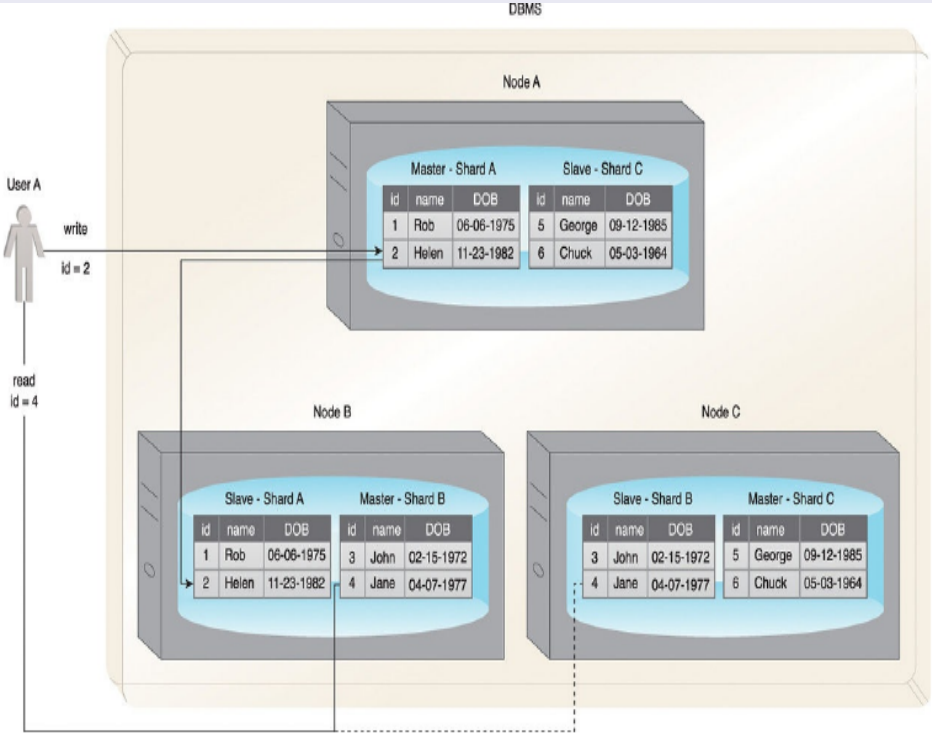

Figure 5.13 An example that shows the combination of sharding and master-slave replication.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

Funcionamiento

Esquema de Shard-Master y Shards-Slaves

Ventajas/Desventajas

- Consistencia de escrituras mantenida por el Shard-Master.
- Si falla el Shard-Master, impacta en las operaciones de escritura.
- Réplicas en Shard-Slaves mejoran escalabilidad y proveen tolerancia a fallos en operaciones de lectura.

NoSQL - Sharding + Réplicas - Peer-to-Peer

Funcionamiento

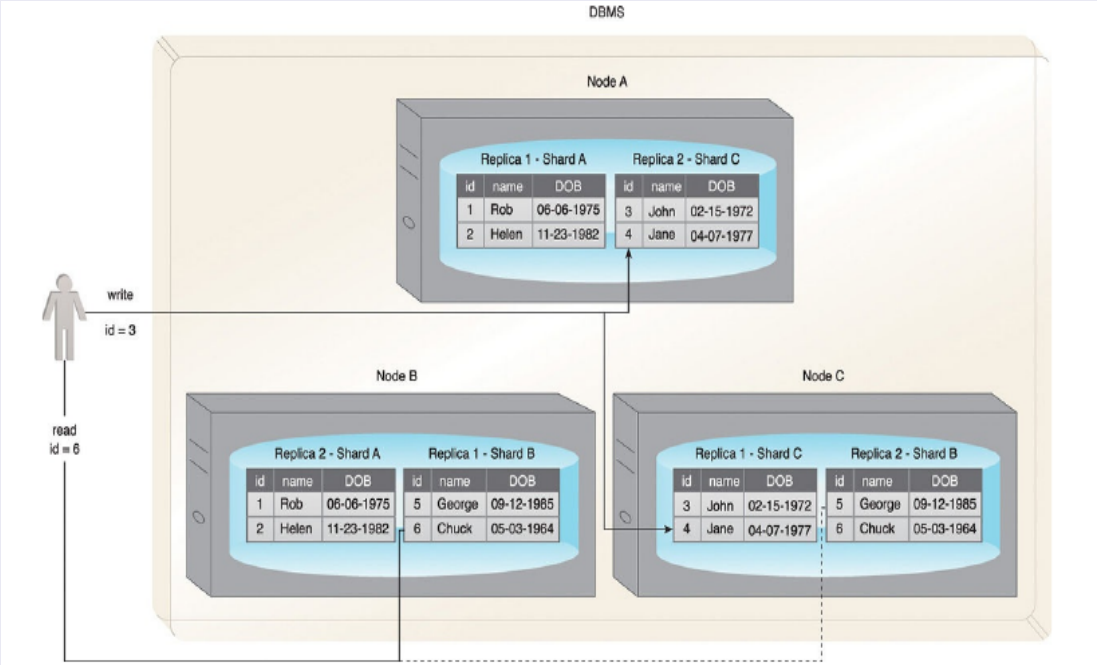

Figure 5.14 An example of the combination of sharding and peer-to-peer replication.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

Funcionamiento

Cada Shard se replica en múltiples peers.

NoSQL - Sharding + Réplicas - Peer-to-Peer

Funcionamiento

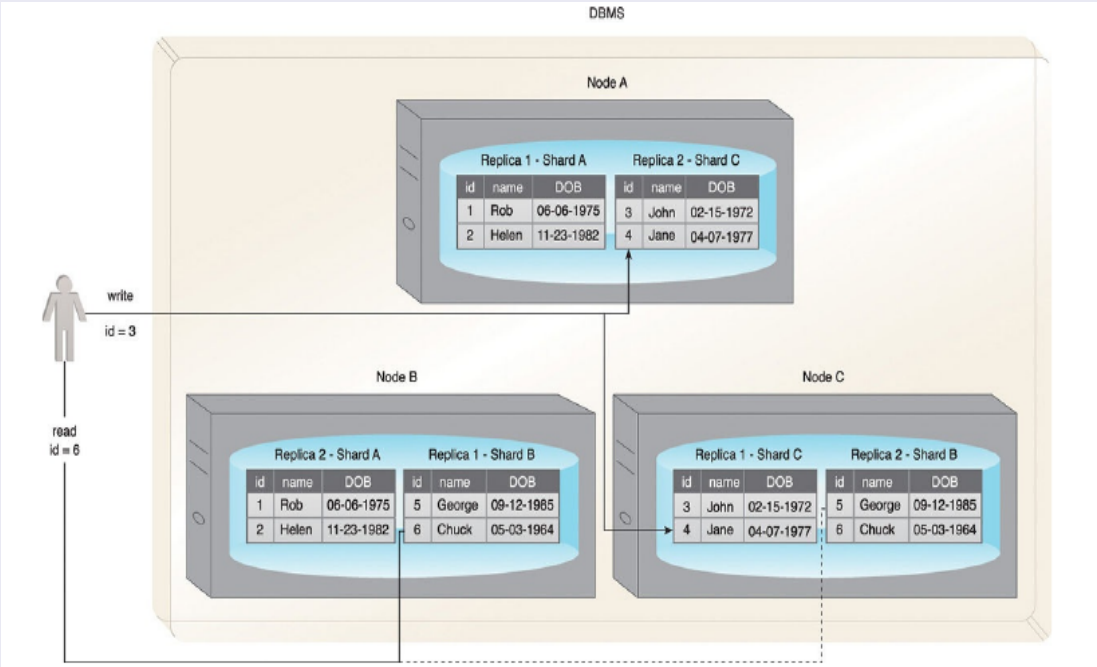

Figure 5.14 An example of the combination of sharding and peer-to-peer replication.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

Funcionamiento

Cada Shard se replica en múltiples peers.

Ventajas/Desventajas

- Este esquema incrementa escalabilidad y tolerancia a fallos.
- Como no hay Master, no existe único punto de falla por lo tanto es tolerante a fallos tanto de operaciones de lectura como escritura.