

MongoDB

José R.R. Viqueira

Centro Singular de Investigación en Tecnoloxías Intelixentes (CITIUS)
Rúa de Jenaro de la Fuente Domínguez,
15782 - Santiago de Compostela.

Despacho: 209

Telf: 881816463

Mail: jrr.viqueira@usc.es

Skype: jrviqueira

URL: <https://citius.gal/team/jose-ramon-rios-viqueira>

Curso 2022/2023

- **Introducción**
- **Modelado de Datos**
- **Replicación**
- **Particionamiento (Sharding)**
- **Consistencia**

Introducción

Modelado

Replicación

Sharding

Consistencia

■ Base de datos de documentos

■ Documento

- ▷ Estructura anidada compuesta de pares (campo, valor)
- ▷ Similar a objetos **JSON**
- ▷ Valores complejos
 - Otros documentos anidados
 - Arrays
 - Arrays de documentos

```
{
  name: "sue",
  age: 26,
  status: "A",
  groups: [ "news", "sports" ]
}
```

← field: value
← field: value
← field: value
← field: value

■ Almacenamiento

- ▷ Colecciones de documentos
- ▷ vistas de solo lectura y vistas materializadas

■ Ventajas

- ▷ Correspondencia con estructuras usadas en lenguajes de programación
- ▷ Reducción de la necesidad de realizar **JOINS**
- ▷ **Esquema dinámico**
 - Proporciona polimorfismo fluido.

Introducción



Modelado

Replicación

Sharding

Consistencia

- Alto rendimiento en la persistencia de datos
 - ▷ Uso de un **modelo anidado**: Permite reducir el coste de E/S
 - ▷ **Indexación**: Permite usar claves de documentos anidados y arrays.
- Lenguaje de consulta expresivo
 - ▷ Operaciones de lectura y escritura (**CRUD**)
 - ▷ **Agregación**
 - ▷ Consultas **de texto completo** y **geoespaciales**.
- Alta disponibilidad
 - ▷ **Replica set**: Grupo de servidores con el mismo conjunto de datos
 - ▷ Recuperación automática ante fallos
 - ▷ Redundancia de datos
- Escalabilidad horizontal
 - ▷ Parte de su funcionalidad principal
 - ▷ El particionamiento (sharding) distribuye los datos entre máquinas
 - ▷ Se pueden crear zonas de datos (usando los valores de la clave)
- Varios motores de almacenamiento

Introducción

Modelado



Replicación

Sharding

Consistencia

- Durante el diseño del modelo de datos, tener en cuenta
 - ▷ Estructura inherente a los propios datos
 - ▷ Forma de acceder a los datos desde las aplicaciones
- **Flexibilidad en el esquema**
 - ▷ Las colecciones de MongoDB no necesitan tener un esquema declarado
 - _ No es necesario que tengan los mismos campos con los mismos tipos de dato.
 - _ Para modificar el esquema (añadir campos, etc.) solo es necesario realizar las modificaciones necesarias en los datos.
 - ▷ Cada documento se puede adaptar así a un objeto concreto, sin que todos los objetos tengan que ser idénticos en estructura
 - ▷ En la práctica los documentos de una colección serán similares
 - _ Se pueden realizar validaciones de esquema si es necesario



■ Estructura de los documentos

- ▷ Decisión de diseño clave
 - Definir la estructura de los documentos y la representación de las relaciones entre datos
- ▷ Datos Anidados
 - Los datos relacionado se almacenan en un único documento de estructura más compleja.
 - Un campo puede tener varias campos dentro o arrays

```
{
  _id: <ObjectId>,
  username: "123xyz",
  contact: {
    phone: "123-456-7890",
    email: "xyz@example.com"
  },
  access: {
    level: 5,
    group: "dev"
  }
}
```

Embedded sub-document

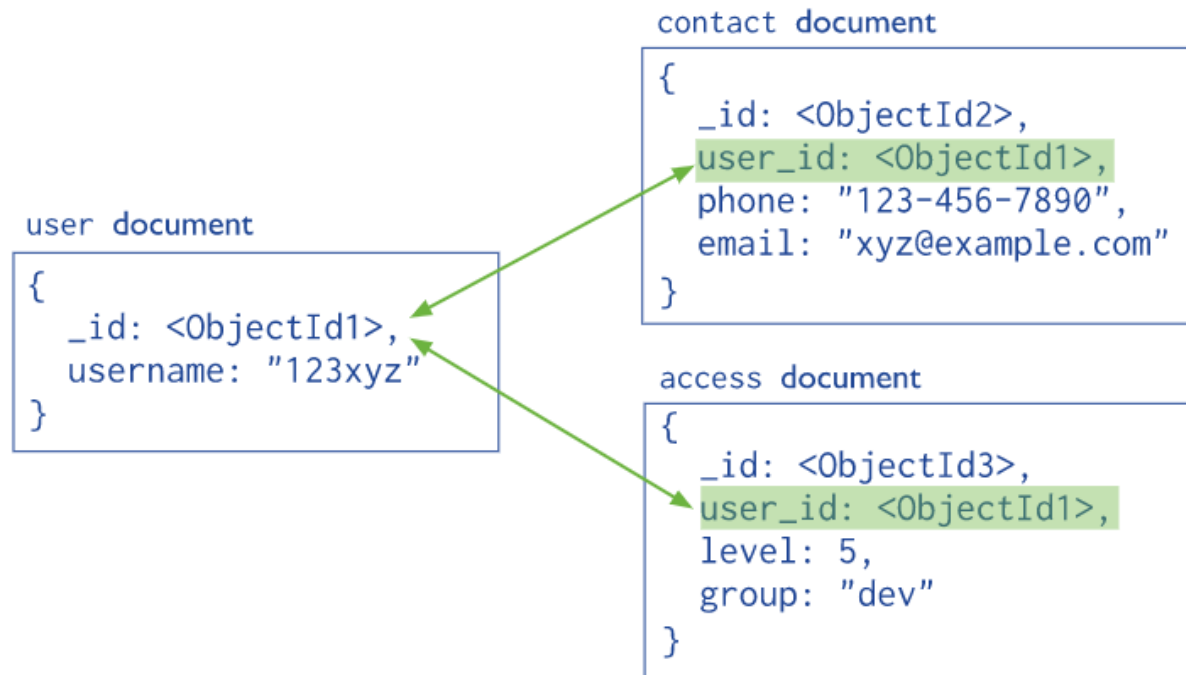
Embedded sub-document



■ Estructura de los documentos

▷ Referencias

- También se pueden almacenar relaciones entre documentos con referencias.





■ Atomicidad y operaciones de escritura

▷ Atomicidad a nivel de documento

- Una operación de escritura es atómica a nivel de cada documento.
- El uso de los agregados simplifica la atomicidad
- Si una operación modifica varios documentos, la operación en su conjunto no es atómica
 - Se pueden entrelazar otras operaciones

▷ Transacciones sobre varios documentos

- Se pueden ejecutar cuando se necesita atomicidad al gestionar (leer y escribir) varios documentos

■ Impacto de la estructura definida en el rendimiento

- ▷ Considerar la forma de acceso durante el diseño.
- ▷ Usar también de forma apropiada la indexación

■ Validación del esquema

- ▷ Se pueden especificar reglas de validación al crear una colección
 - Se pueden aplicar de forma más o menos estricta y también generar errores o avisos (warnings)
- ▷ Soporte para JSON Schema desde la versión 3.6

Introducción

Modelado

Replicación

Sharding

Consistencia

■ Introducción

- ▷ *Replica set*: grupo de procesos **mongod** que mantienen el mismo conjunto de datos.
- ▷ Proporciona **redundancia** y **alta disponibilidad**



■ Redundancia y disponibilidad de los datos

- ▷ Varias copias de los mismos datos en servidores distintos
 - _ Tolerancia ante el fallo de algún servidor
- ▷ Puede mejorar el rendimiento de algunas lecturas
- ▷ Se puede mejorar la localidad de los datos y la disponibilidad de los mismos

■ Replicación en MongoDB

▷ En cada **replica set**

- Un nodo primario
- Varios nodos secundarios
- Opcionalmente un nodo arbiter

▷ **Nodo primario**

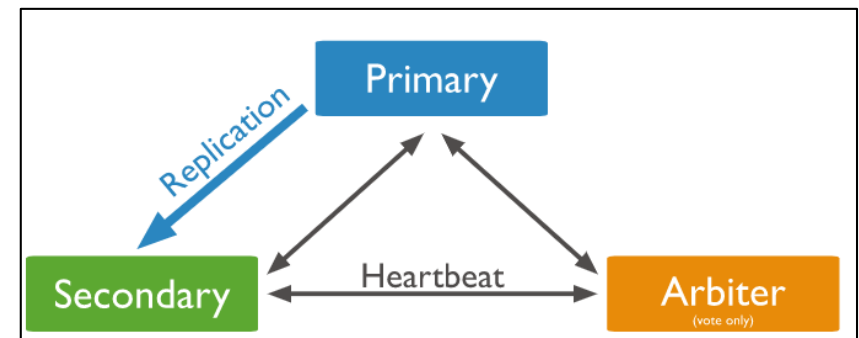
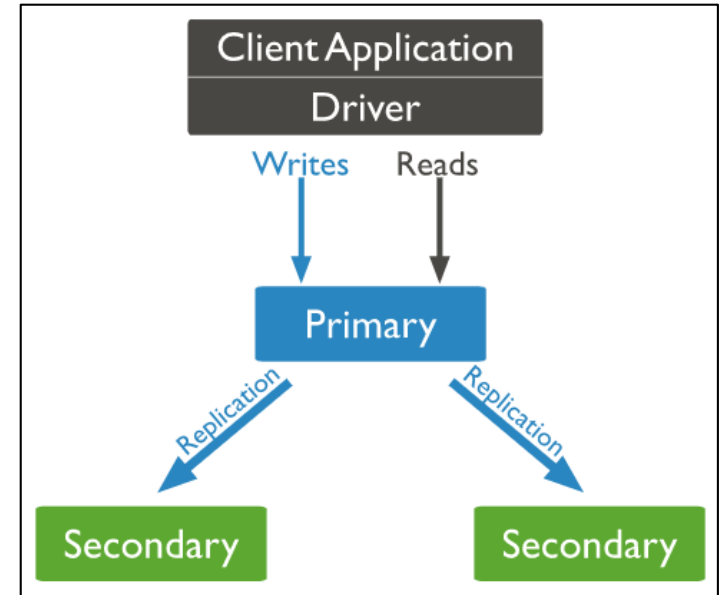
- Recibe todas las escrituras

▷ **Nodos Secundarios**

- Replican los cambios que se van produciendo en el primario
- Si el primario falla, uno de los secundarios puede ser elegido como primario

▷ **Nodo Arbiter**

- Participa en las elecciones pero no tiene datos
- No pueden transformarse en secundarios o primarios



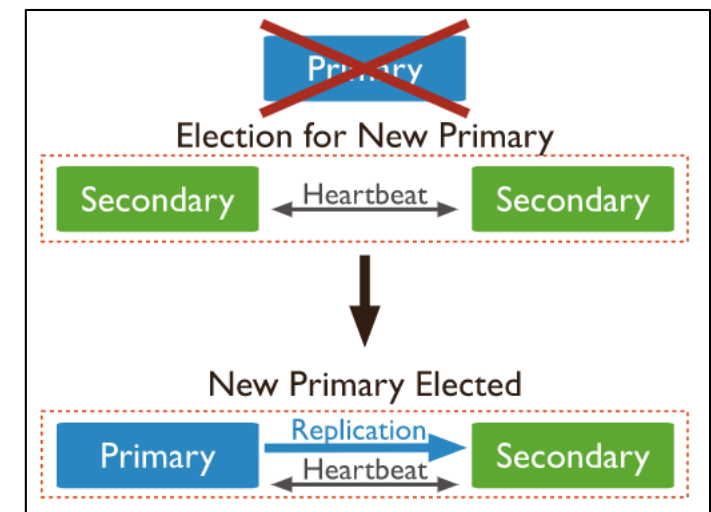


■ Replicación asíncrona

- ▷ Los nodos secundarios replican los cambios de los primarios de forma asíncrona.
- ▷ Existe cierto retraso entre que un cambio se hace en el primario y se replica en todos los secundarios
 - _ Para limitar este retraso se puede limitar el ratio de modificaciones que acepta el primario

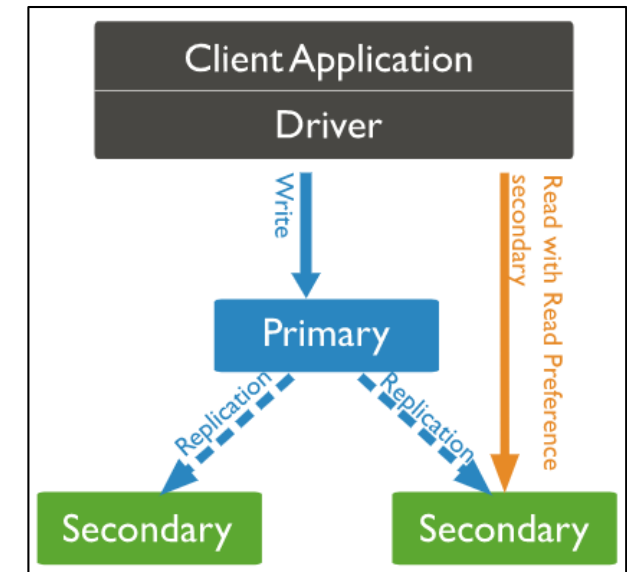
■ Recuperación automática ante fallos

- ▷ Cuando el primario no responde durante cierto tiempo, los secundarios eligen otro primario



■ Operaciones de lectura

- ▷ Por defecto se realizan del primario.
- ▷ Los clientes pueden indicar como preferencia de lectura hacerlo desde un secundario
- ▷ Los secundarios pueden no tener los últimos cambios del primario
- ▷ Las transacciones multi-documento deben de leer preferentemente del primario. Todas las operaciones deben de guiarse hacia el mismo miembro.
- ▷ Dependiendo del compromiso de lectura usado, los clientes pueden ver cambios antes de que estén confirmado (durables).
 - _ Si se usa el compromiso "local" o "available" se pueden leer datos con cambios que pueden todavía deshacerse.



Introducción

Modelado

Replicación



Sharding

Consistencia

■ Transacciones

- ▷ La preferencia de lectura debe de estar en el primario. Todas las operaciones deben de dirigirse al mismo nodo
- ▷ Los cambios no serán visibles hasta que se compromete
- ▷ Si se escriben varias particiones (shards), otro lector podría ver los cambios en una y no los cambios en otra.

■ Otras características

- ▷ Suscripción a flujos de cambios
- ▷ Se puede controlar la forma de elegir primarios
- ▷ Se pueden desplegar miembros en centros de datos distintos
- ▷ Se pueden dedicar miembros para backup, etc.

Introducción

Modelado

Replicación

Sharding



Consistencia

■ Introducción

- ▷ Distribución de los datos entre varias máquinas
 - _ Gran volumen de datos
 - _ Muchas operaciones por segundo
- ▷ **Escalamiento vertical**
 - _ Incrementar la capacidad de un único servidor (CPU, RAM, etc.).
 - _ Límite en las capacidades del hardware actual.
- ▷ **Escalamiento horizontal**
 - _ Particionar los conjuntos de datos para ser repartidos entre múltiples servidores.
 - _ Añadir nuevos servidores es en general más sencillo que mejorar el hardware de un servidor.
 - _ La contrapartida es la mayor complejidad de la infraestructura y del mantenimiento

Introducción

Modelado

Replicación

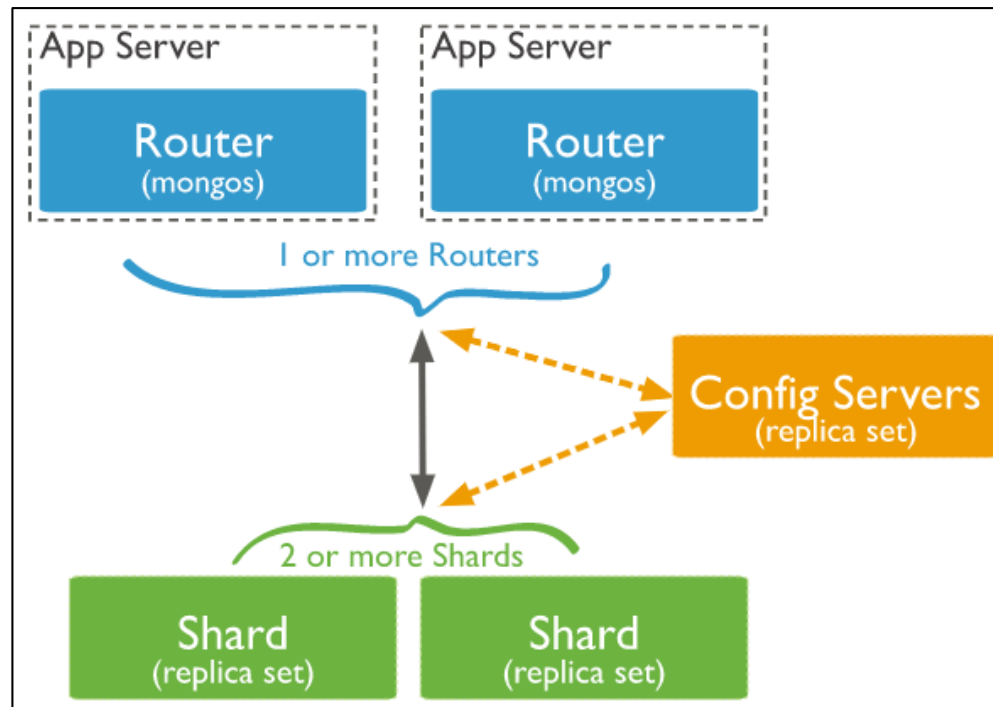
Sharding

Consistencia

Cluster particionado (Sharded Cluster) en MongoDB

► Componentes

- **Shards**: Puede ser un simple proceso o un replica set. Contiene los datos de una partición
- **mongos**: Enrutador de consultas. Interfaz entre los clientes y el cluster.
- **config server**: Almacenan los metadatos y la información de configuración del cluster



Introducción

Modelado

Replicación

Sharding

Consistencia

- Clave de particionamiento (Shard key)
 - ▷ Un campo o varios campos del documento
 - ▷ Determina como se distribuyen los documentos en las particiones (shards)
 - ▷ Debe de existir un índice que empieza por la clave de particionamiento
 - ▷ La elección de la clave afecta mucho al rendimiento
- Chunks
 - ▷ Los datos se particionan en chunks.
- Balanceador
 - ▷ Para lograr una distribución uniforme de los chunks en el cluster un balanceador se ejecuta en segundo plano para mover chunks entre los shards
- Ventajas del particionamiento
 - ▷ Distribución de lecturas y escrituras entre los shards. Consultas sobre la clave se pueden dirigir a shards concretos
 - ▷ El espacio de almacenamiento se incrementa añadiendo nuevos shards
 - ▷ Alta disponibilidad al usar replica sets en cada componente (shards, config)

Particionamiento (Sharding)

Introducción

Modelado

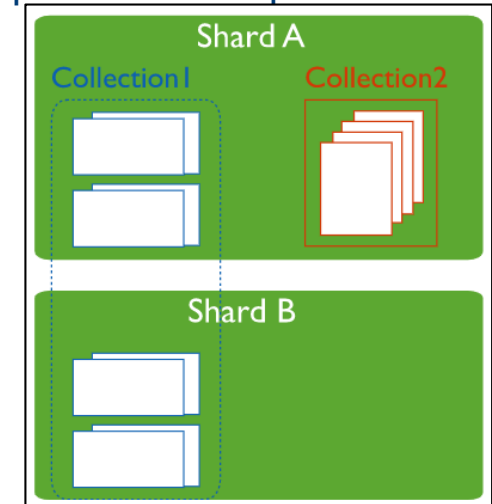
Replicación

Sharding

Consistencia

■ Consideraciones antes del particionamiento

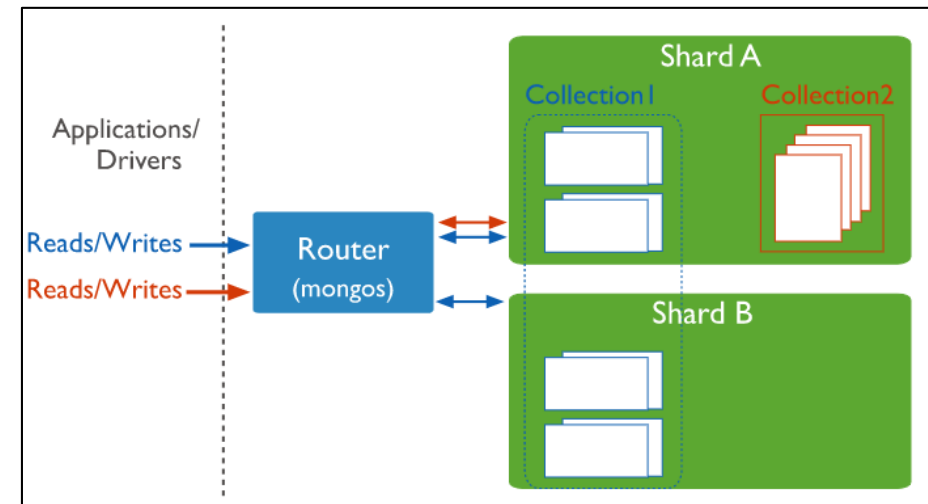
- ▷ La complejidad de la infraestructura demanda cierta planificación para la ejecución y el mantenimiento
- ▷ Una vez se particiona, no se puede desparticionar
- ▷ Hay que analizar con cuidado cual es la clave de particionamiento



■ Una base de datos puede tener colecciones particionadas y no particionadas

- ▷ Las no particionadas se almacenan en el shard primario

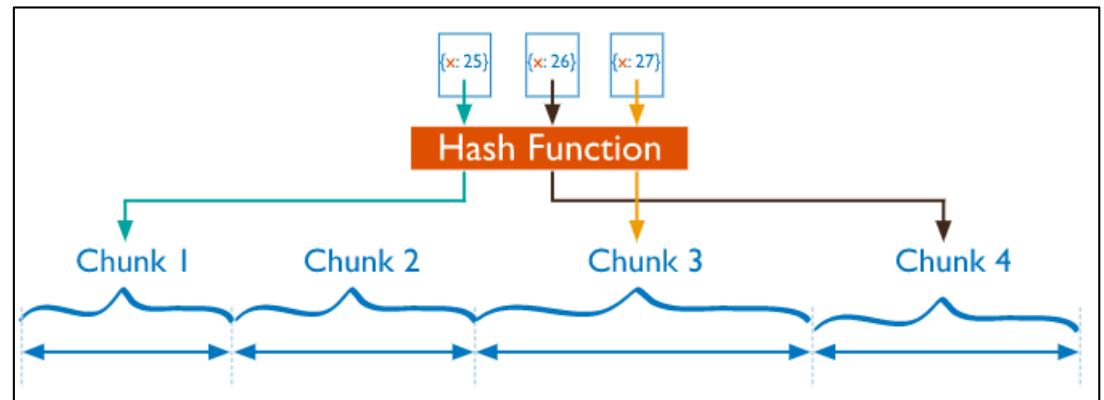
■ La conexión con un cluster particionado se hace a través de **mongos**



■ Estrategia de particionamiento

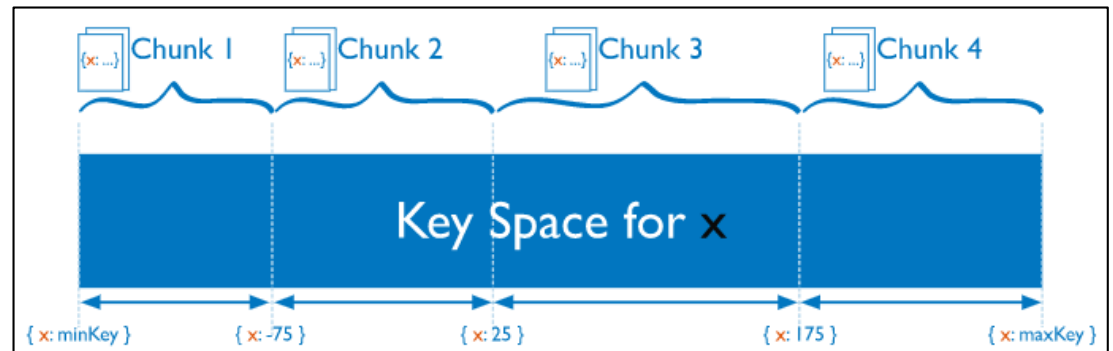
▷ Hashed Sharding

- Se evalúa una función hash sobre la clave. Cada chunk almacena documentos para un rango de valores resultantes del hash.
- Consiguen buena distribución sobre todo cuando se aplican sobre claves que cambian monótonicamente.
- Las búsquedas por rango pueden necesitar acceder va muchos chunks



▷ Ranged Sharding

- Los chunks se construyen sobre rangos de la clave
- Las búsquedas por rango se pueden dirigir a shards concretos
- La elección de la clave es muy importante para conseguir una buena distribución



Introducción

Modelado

Replicación

Sharding



Consistencia

■ Elección de la clave

▷ Cardinalidad

- _ Si elegimos una clave con pocos valores diferentes tendremos el problema de que el cluster podrá tener pocos shards

▷ Frecuencia

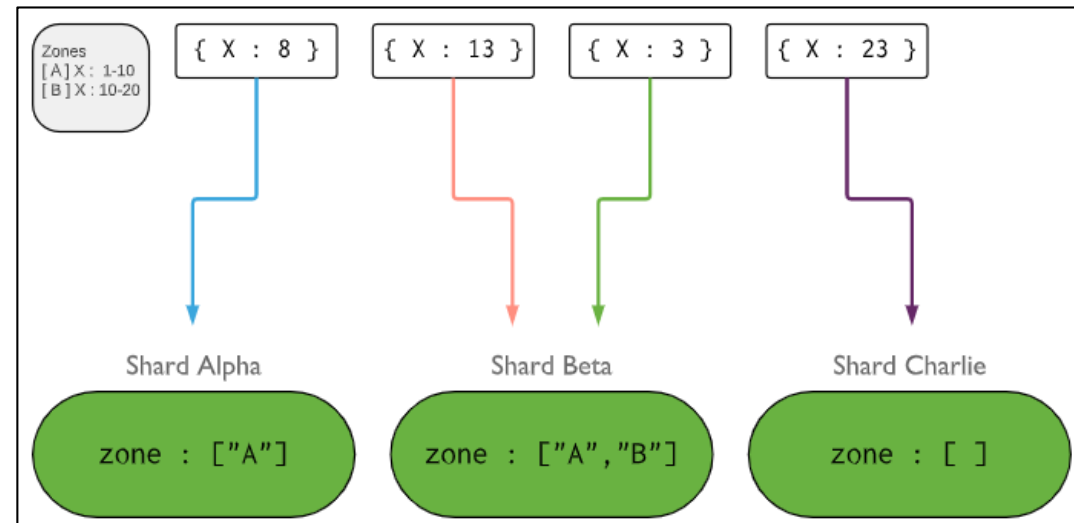
- _ La distribución de los valores de la clave influye.
- _ Si los valores están muy concentrados
 - los datos se almacenarían en unos pocos chunks
 - los accesos a los shards no estarían bien distribuidos, que es lo que se busca
- _ Lo ideal sería tener una distribución uniforme de las claves

▷ Monotonicidad

- _ Si la clave crece o decrece de forma monotónica
 - la mayoría de documentos irán a shards que almacenan valores extremos

■ Zonas en clusters particionados

- ▷ Cuando un cluster se expande por varios centros de datos las **zonas** pueden ayudar a mejorar la localidad de los datos
- ▷ Las zonas se crean en función de la clave de particionamiento
- ▷ Cada zona se puede asociar con uno o varias particiones (shards) del cluster
- ▷ El balanceador mueve chunks de una zona solo a shards de la misma zona



Introducción

Modelado

Replicación

Sharding

Consistencia



■ Preferencias de lectura y compromisos de lectura y escritura

- ▷ Impacto en el rendimiento
- ▷ Impacto en la consistencia
- ▷ Compromiso entre consistencia y disponibilidad

■ Preferencias de lectura

- ▷ <https://docs.mongodb.com/manual/core/read-preference/>
- ▷ Se puede especificar en la conexión y cambiar en la Shell
 - `db.getMongo().setReadPref('nearest')`
- ▷ Opciones
 - **primary** (opción por defecto): Se lee del nodo primario del replica set.
 - **primaryPreferred**: Intenta en el primario y si no puede lee del secundario
 - **secondary**: Se lee de un nodo secundario.
 - **secondaryPreferred**: Intenta en el secundario, y si no puede lee del primario
 - **nearest**: Lee de cualquier nodo intentando minimizar un cálculo de latencia previo

Introducción

Modelado

Replicación

Sharding

Consistencia



■ Consistencia causal

- ▷ <https://docs.mongodb.com/manual/core/read-isolation-consistency-recency/#>
- ▷ Las operaciones futuras ven el efecto de operaciones pasadas.
 - Ejemplo: Después de hacer un borrado, no debería de poder leer el dato borrado.
- ▷ En MongoDB se puede conseguir dentro de una sesión de usuario.
 - Usando compromiso "majority" en lecturas y escrituras.
- ▷ Garantías proporcionadas
 - **Read your writes**: Lectura posterior leer el valor actualizado por una escritura anterior.
 - **Monotonic reads**: Lectura no puede obtener estado previo a una lectura anterior.
 - **Monotonic writes**: Las escrituras se ejecutan en orden.
 - **Writes follow reads**: Las escrituras que se solicitan después de la lectura se ejecutan en ese orden
- ▷ No se proporciona aislamiento entre las operaciones de una sesión y otras sesiones de usuario.

Introducción

Modelado

Replicación

Sharding

Consistencia



■ Compromiso de lectura (Read Concern)

- ▷ <https://docs.mongodb.com/manual/reference/read-concern/>
- ▷ Se puede especificar en las operaciones de lectura (find, count, distinct, etc.)
 - `db.collection.find().readConcern('majority')`
- ▷ Niveles
 - **local** (defecto en lecturas de primario en sesiones de consistencia causal)
 - No se verifica que se haya escrito en la mayoría de nodos.
 - **available** (defecto en lecturas de secundario fuera de consistencia causal)
 - No se verifica escritura en mayoría de nodos. Latencia menor cuando hay sharding (solo cuando hay sharding se diferencia del nivel "local". No espera a tener consistencia en la particiones.
 - **majority**: Lee datos que han sido comprometidos en la mayoría de replicas. Datos leídos son durables.
 - **linearizable**: Obtiene datos comprometidos. Puede tener que esperar por escrituras en curso. No se puede usar en transacciones. Solo en lecturas del primario.
 - **snapshot**: Solo se puede usar con transacciones. Lee datos comprometidos por la mayoría.

Introducción

Modelado

Replicación

Sharding

Consistencia



■ Compromiso de escritura (Write Concern)

- ▷ <https://docs.mongodb.com/manual/reference/write-concern/>
- ▷ Se especifica usando una estructura los siguientes campos
 - **w**: Nivel de compromiso de escritura
 - **w:1** Se necesita confirmación de escritura en el primario. Las escrituras en los secundarios se hacen después de forma asíncrona
 - **w:0** No requiere confirmación de escritura alguna.
 - **w:número mayor que 1** Confirmación en el primario y varios secundarios (w:2 confirma el primario y un secundario)
 - **w:majority** Confirmación de la mayoría de nodos
 - **w:nodo** Confirmación de un nodo concreto.
 - **j**: Confirmación de que se ha escrito en el on-disk journal (registro histórico o log de MongoDB)
 - **wtimeout**: Solo cuando w>1. Si se sobrepasa el tiempo indicado se genera un error.
- ▷ Se puede especificar a nivel de transacción o a nivel de operación. Se puede especificar de forma global

```
db.adminCommand(
{
  setDefaultRWConcern : 1,
  defaultReadConcern: { <read concern> },
  defaultWriteConcern: { <write concern> },
  writeConcern: { <write concern> },
  comment: <any>
})sd
```


MongoDB

<https://docs.mongodb.com/manual/>

José R.R. Viqueira

Centro Singular de Investigación en Tecnoloxías Intelixentes (CITIUS)
Rúa de Jenaro de la Fuente Domínguez,
15782 - Santiago de Compostela.

Despacho: 209

Telf: 881816463

Mail: jrr.viqueira@usc.es

Skype: jrviqueira

URL: <https://citius.gal/team/jose-ramon-rios-viqueira>

Curso 2022/2023