`mongodb`分片

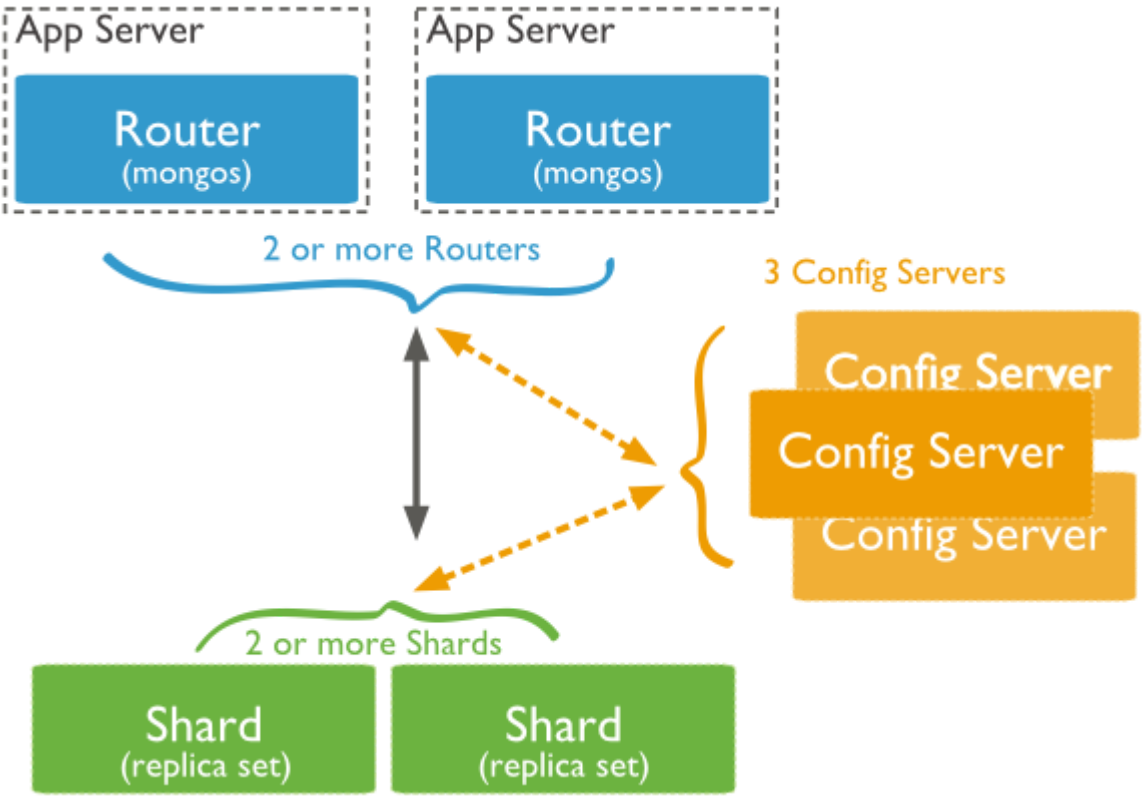
概念

当MongoDB存储海量的数据时，一台机器可能不足以存储数据，也可能不足以提供可接受的读写吞吐量。这时，我们就可以通过在多台机器上分割数据，使得数据库系统能存储和处理更多的数据。

为什么使用分片

- 复制所有的节点写入操作到节点
- 延迟的敏感数据会在主节点查询
- 单个副本集限制在12个节点
- 当请求量巨大时会出现内存不足
- 本地磁盘不足
- 垂直扩展价格昂贵

分片组件



- **Shard**

  用于存储实际的数据块，实际生产环境中一个`shard server`角色可由几台机器组个一个`replica set`承担，防止主机单点故障

- **Config Server:**

`mongod`实例，存储了整个`ClusterMetadata`，其中包括`chunk`信息，即数据的哪一部分放在哪一个`shard`上，`router`将会利用这些元数据将请求分发到对应的`shards`上，`shards`上`chunk`的迁移也是`config server`来控制的

- **Query Routers:**

  前端路由，客户端由此接入，且让整个集群看上去像单一数据库，前端应用可以透明使用。

数据分割（data partition）

`MongoDB`在`collection`这个级别进行数据的切块，称之为`sharding`。块的最小粒度是`chunk`，其大小（`chunkSize`）默认为64M。

当一个集合的数据量超过`chunkSize`的时候，就会被拆分成两个`chunk`，这个过程称为`splitting`。那么按照什么原则将一个`chunk`上的数据拆分成两个`chunk`，这就是`Sharding key`的作用，`Sharding key`是被索引的字段，通过`sharding key`，就可以把数据均分到两个`chunk`，每一个`document`在哪一个`chunk`上，这就是元数据信息。元数据信息存放在`config server`上，方便`router`使用。

如果`sharding cluster`中有多个`shard`，那么不同`shard`上的`chunk`数目可能是不一致的，这个时候会有一个后台进程（`balancer`）来迁移（`migrate`）`chunk`，从`chunk`数目最多的`shard`迁移到`chunk`数目最少的`chunk`，直到达到均衡的状态。迁移的过程对应用程序来说是透明的。

如何选择片键

1. 对集合进行分片时，要选择一或两个字段用于拆分数据，这个键就叫做片键。

2. 拆分数据最常用的数据分发方式有三种：升序片键、随机分发的片键和基于位置的片键。

1). 升序片键：升序片键通常有点类似于"`date`"字段或者是`ObjectId`，是一种随着时间稳定增长的字段。缺点：例如`ObjectId`可能会导致接下来的所有的写入操作都在同一块分片上。

2). 随机分发的片键：随机分发的片键可以是用户名，邮件地址，`UDID`，`MD5`散列值或者数据集中其他一些没有规律的键。缺点：`MongoDB`在随机访问超出`RAM`大小的数据时效率不高。

3). 基于位置的片键：基于位置的片键可以是用户的IP、经纬度、或者地址。这里的"位置"比较抽象，不必与实际的物理位置字段相关。

如果希望特定范围内的块出现在特定的分片中，可以为分片添加tag，然后为块指定相应的tag

3. 片键策略：

1). 散列片键：如果追求的是数据加载速度的极致，那么散列片键是最佳选择。散列片键可使其他任何键随机分发，因此，如果打算在大量查询中使用使用升序键，但同时又希望写入数据随机分发的话，

散列片键会是一个非常好的选择。缺点：无法使用散列片键做指定目标的范围查找。

创建步骤：

```
db.users.ensureIndex({"username":"hashed"});
sh.shardCollection("app.users",{"username":"hashed"})
```

2). `GridFS` 的散列片键

3). 流水策略：如果有一些服务器比其他服务器更强大，我们可能希望让这些强大的服务器处理更多的负载。比如说：加入有一个使用SSD的分片能够处理10倍于其他机器的负载。我们可以强制将所有新数据

插入到SSD，然后让均衡器将旧的块移动到其他分片上。

a. 为SSD指定一个标签：

```
sh.addShardTag("shard-name","ssd")
```

b. 将升序键的当前值一直到正无穷范围的块都指定分布在SSD分片上：

```
sh.addTagRange("dbName.collName",{"_id":ObjectId()},...{"_id":MaxKey},"ssd")
```

所有插入请求均会路由到这个块上，这个块始终位于标签的 `ssd` 的分片上。

c. 除非修改标签范围，否则从升序键的当前值一直到正无穷都被固定在这个分片上。可以创建一个定时任务每天更新一次标签范围：

```
use config
var tag =db.tags.findOne({"ns":"dbName.collName",..."max":{"shardKey":MaxKey}})
tag.min.shardKey = ObjectId()
db.tags.save(tag)
```

这样前一天的数据就会被移动到其他分片上了。

此策略的另一个缺点：需要修改才能进行扩展。如果写请求超出了SSD的处理能力，无法进行负载均衡。

4). 多热点：写请求分布在集群中时，分片是最高效的。这种技术会创建多个热点(最好在每个分片上都创建几个热点)，写请求于是会均衡地分布在集群内，而在单个分片上则是以升序分布的。

为了实现这种方式，需使用复合片键。复合片键中的第一个值只是比较粗略的随机值，势也比较低。

4. 片键规则和指导方针：

1). 片键限制：片键不可以是数组。文档一旦插入，其片键就无法修改了。要修改文档的片键值，就必须先删除文档。

2). 片键的势：选择一个值会变化的的键非常重要，即值很多，随着数据量的增大可以分出更多的片键。分片在势比较高的字段上性能更佳。

5. 控制数据分发

1). 对多个数据库和集合使用一个集群：通过 `tag` 标记，将重要的数据放到性能更好的服务器上，将不重要的数据放在性能一般的服务器上。

2). 手动分片：如果不希望数据被自动分发，可以关闭均衡器，使用 `moveChunk` 命令手动对数据进行迁移。

分片管理

1. 检查集群状态：

1). 使用 `sh.status` 查看集群摘要信息: 块的数量比较多时， `sh.status()` 命令会概述块的状态，而非打印出每个块的相关信息。如需查看所有的块，可使用 `sh.status(true)` 命令。

`sh.status()` 显示的所有信息都来自 `config` 数据库。运行 `sh.status()` 命令，使用MapReduce获取这一数据。因此，如果启动数据库时指定-- `noscripting` 选项，则无法运行 `sh.status()` 命令。

2). 检查配置信息：

a. 集群相关的所有配置信息都保存在配置服务器上 `config` 数据库的集合中。可以直接访问该数据库，不过 `shell` 提供了一些辅助函数。

b. 永远不要直接连接到配置服务器，以防止配置服务器数据被不小心修改或删除。应该先连接到 `mongos` ，然后通过 `config` 数据库来查询相关信息： `use config`

如果通过 `mongos` 操作配置数据， `mongos` 会保证将修改同步到所有配置服务器，也会防止危险的操作发生，如意外删除 `config` 数据库等。

c. 总的来说，不应直接修改 `config` 数据库中的任何数据。如果确实修改了某些数据，通常需要重启所有的 `mongos` 服务器，才能看到效果。

d. config中几个关键集合：

shards : 跟踪记录集群中所有分片的信息。

databases: 跟踪记录集群中所有数据库的信息，不管数据库有没有分片。

collections: 跟踪记录所有分片集合的信息(非分片集合信息除外)

chunks: 记录集合中所有块的信息。

changelog: 跟踪记录集群的操作，因为该集合会记录所有拆分和迁移的操作。

tags: 该集合的创建是在为系统配置分片标签时发生的。每个标签都与一个块范围相关联。

settings: 该集合含有当前的均衡器设置和块大小的文档信息。通过修改该集合的文档，可开启和关闭均衡器，也可以修改块的大小。注意，应总是连接到mongos修改该集合的值。

2. 查看网络连接：

1). 查看连接统计：可以使用 `connPoolStats` 命令，查看 `mongos` 和 `mongod` 之间的连接信息：

```
db.adminCommand({"connPoolStats":1})
```

在一个分片上执行 `connPoolStats` ，输出信息中可以看到该分片与其他分片间的连接，包括连接到其他分片做数据迁移的连接。

2). 限制连接数量： 可在`mongos`的命令行配置中使用`maxConns`选项，这样可以限制`mongos`能够创建的连接数量。可以使用下面公式计算分片能够处理的来自单一mongos连接数量：

```
    maxConns = 20000 - (mongos进程的数量 * 3 ) - (每个副本集的成员数量 * 3 ) - (其他/mongos进程的数量)
```

> **MongoDB如果没有安全退出，那些已经打开的套接字很可能没有被关闭。**
>
> **在出现大量重新连接时，除了重启进程，没有其他特殊有效的方法。**

3. 服务器管理

1). 添加服务器：使用`addShard`命令，向集群中添加新的分片

2). 修改分片的服务器：要修改分片的成员，需直接连接到分片的主服务器上，然后对副本集进行重新配置。集群配置会自动检测更改，并将其更新到`config.shards`上。

3). 通常来说，不应从集群中删除分片。执行`removeShard`命令排除数据和查看排出进度。

4). 修改配置服务器：修改配置服务器非常困难，而且有风险，通常还需要停机。注意，修改配置服务器前，应做好备份。

首先必须关闭所有`mongos`进程，然后使用新的--`configdb`参数重启所有mongos进程。

4. 数据均衡：

1). 均衡器：均衡器只使用块的数量，而非数据大小，作为衡量分片间是否均衡的指标。自动均衡总是根据数据集的当前状态来决定数据迁移，而不考虑数据集历史状态。我们可以手动均衡数据集块的数量。

2). 修改块的大小：块的大小默认为64M，这个大小的块既易于迁移，又不至于导致过多的流失。使用shell连接到`mongos`，修改`config.setting`集合，从而完成块大小的修改。

该设置的有效范围是整个集群：它会影响所有集合的数据库。因此，如需对一个集合使用较小的块，而对另一个集合使用较大的块，比较好的解决方式是取一个折中值(或者将这两个值放到不同的集合中)。

如果`MongoDB`频繁进行数据迁移或文档增大，则可能需要增加块的大小。

3). 迁移块：同一块内的所有数据都位于同一分片上。如该分片的块数量比其他分片多，则MongoDB会将其中的一部分块迁移到其他块数量较少的分片上。移动快的过程叫迁移，MongoDB就是这样在集群中

实现数据均衡的。可在`shell`中使用`moveChunk`辅助函数，手动移动块。

如果某个块的大小超出了系统指定的最大值，`mongos`则会拒绝移动这个块。移动之前必须先手动拆分这个块，可以使用`splitAt`命令对块进行拆分。特大块，无法被拆分。

4). 特大块：某些片键，值比较少，例如：日期等。可能会形成超出设置的最大块大小的块，这种块成为特大块.

出现特大块的表现之一是，某个分片的大小增长速度要比其他分片块的多。也可使用`sh.status()`来检查是否出现了特大块；特大块会存在一个`jumbo`属性。

a. 分发特大块，一个复杂的过程

b. 防止特大块的出现：修改片键，细化片键的粒度

5). mongos有时无法从配置服务器正确更新配置。如果发现配置有误，`mongos`的配置过旧或无法找到应有的数据，可以使用`flushRouterConfig`命令手动刷新所有缓存：

```
db.adminCommand({"flushRouterConfig":1})
```

如`flushRouterConfig`命令没能解决问题，则应重启所有的`mongos`或者`mongod`进程，以便清除所有可能的缓存。