MongoDB 分片集群技术

分类: NoSQL,其他

在了解分片集群之前，务必要先了解复制集技术！

1.1 MongoDB复制集

一组Mongodb复制集，就是一组mongod进程，这些进程维护同一个数据集合。复制集提供了数据冗余和高等级的可靠性，这是生产部署的基础。

1.1.1 复制集的目的

保证数据在生产部署时的冗余和可靠性，通过在不同的机器上保存副本来保证数据的不会因为单点损坏而丢失。能够随时应对数据丢失、机器损坏带来的风险。

换一句话来说，还能提高读取能力，用户的读取服务器和写入服务器在不同的地方，而且，由不同的服务器为不同的用户提供服务，提高整个系统的负载。

是一组
一个数据
集，实例可以在不同的机器上面。实例中包含一个主导，接

受客户端所有的写入操作，其他都是副本实例，从主服务器上获得数据并保持同步。

主服务器很重要，包含了所有的改变操作（写）的日志。但是副本服务器集群包含有所有的主服务器数据，因此当主服务器挂掉了，就会在副本服务器上重新选取一个成为主服务器。

每个复制集还有一个仲裁者，仲裁者不存储数据，只是负责通过心跳包来确认集群中集合的数量，并在主服务器选举的时候作为仲裁决定结果。

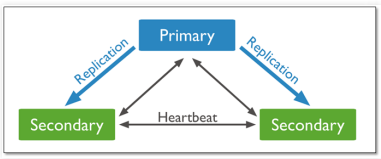
1.2 复制的基本架构

基本的架构由3台服务器组成，一个三成员的复制集，由三个有数据，或者两个有数据，一个作为仲裁者。

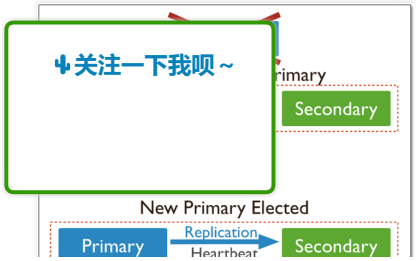
1.2.1 三个存储数据的复制集

具有三个存储数据的成员的复制集有：

> *一个主库；*
> *两个从库组成，主库宕机时，这两个从库都可以被选为主库。*



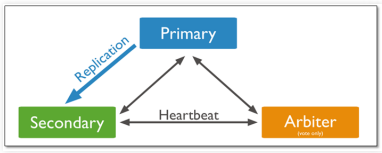
当主库宕机后,两个从库都会进行竞选，其中一个变为主库，当原主库恢复后，作为从库加入当前的复制集群即可。



1.2.2 当存在arbiter节点

在三个成员的复制集中，有两个正常的主从，及一台arbiter节点：

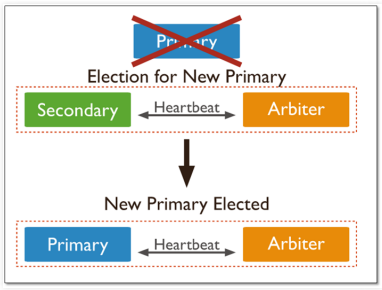
> *一个主库*
> *一个从库，可以在选举中成为主库*
> *一个aribiter节点，在选举中，只进行投票，不能成为主库*



说明：

> *由于arbiter节点没有复制数据，因此这个架构中仅提供一个完整的数据副本。arbiter节点只需要更少的资源，代价是更有限的冗余和容错。*

当主库宕机时，将会选择从库成为主，主库修复后，将其加入到现有的复制集群中即可。



1.2.3 Primary选举

SetInitiate
ell的
始化，初始
发送心跳
消息，并发起Priamry选举操
作，获得『大多数』成员投票

支持的节点，会成为Primary，其余节点成为Secondary。

『大多数』的定义

假设复制集内投票成员（后续介绍）数量为N，则大多数为 N/2 + 1，当复制集内存活成员数量不足大多数时，整个复制集将无法选举出Primary，复制集将无法提供写服务，处于只读状态。

投票成员数	大多数	容忍失效数
1	1	0
2	2	0
3	2	1
4	3	1
5	3	2
6	4	2
7	4	3

通常建议将复制集成员数量设置为奇数，从上表可以看出3个节点和4个节点的复制集都只能容忍1个节点失效，从『服务可用性』的角度看，其效果是一样的。（但无疑4个节点能提供更可靠的数据存储）

1.3 复制集中成

1.3.1 所有成员说明

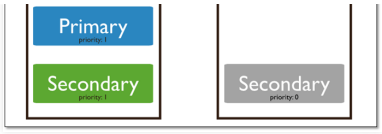
成员	说明
	正常情况下，复制集的Seconary会参与Primary选举（自身也可能会被选为Primary），并从步最新写以保证与储相同的
Second	Secondary可以

ary	提供读服务，增加Secondary节点可以提供复制集的读服务能力，同时提升复制集的可用性。另外，Mongodb支持对复制集的Secondary节点进行灵活的配置，以适应多种场景的需求。
Arbiter	Arbiter节点只参与投票，不能被选为Primary，并且不从Primary同步数据。 比如你部署了一个2个节点的复制集，1个Primary，1个Secondary，任意节点宕机，复制集将不能提供服务了（无法选出Primary），这时可以给复制集添加一个Arbiter节点，即使有节点宕机，仍能选出Primary。 Arbiter本身不存储数据，是非常轻量级的服务，当复制集成员为偶数时，最好加入一个Arbiter节点，以提升复制集可用性。
Prio	Priority0节点的选举优先级为0，不会被选举为Primary 比如你跨机房A、B部署了一个复制集，并且想指定Primary必须在A机房，这时可以将B机房的复制集成员 置为0，这 就一定会 成员。 如果这样部署，最好将『大多

	数』节点部署在A机房，否则网络分区时可能无法选出Primary）
Vote 0	Mongodb 3.0里，复制集成员最多50个，参与Primary选举投票的成员最多7个，其他成员（Vote0）的vote属性必须设置为0，即不参与投票。
Hidden	Hidden节点不能被选为主（Priority为0），并且对Driver不可见。因Hidden节点不会接受Driver的请求，可使用Hidden节点做一些数据备份、离线计算的任务，不会影响复制集的服务。
Delayed	Delayed节点必须是Hidden节点，并且其数据落后与Primary一段时间（可配置，比如1个小时）。 因Delayed节点的数据比Primary落后一段时间，当错误或者无效的数据写入Primary时，可通过Delayed节点的数据来恢复到之前的时间点。

1.3.2 Priority 0节点

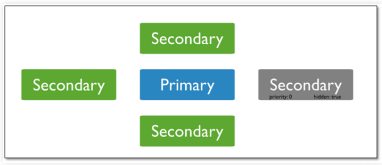
作为一个辅助可以作为一个备用。在一些复制集中，可能无法在合理的时间内添加新 成员保持数 够替换不



1.3.3 Hidden 节点（隐藏节点）

客户端将不会把读请求分发到隐藏节点上，即使我们设定了 复制集读选项 。

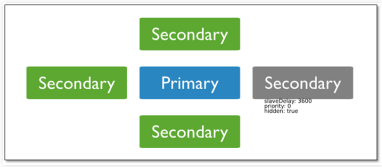
这些隐藏节点将不会收到来自应用程序的请求。我们可以将隐藏节点专用于报表节点或是备份节点。 延时节点也应该是一个隐藏节点。



1.3.4 Delayed 节点（延时节点）

延时节点的数据集是延时的，因此它可以帮助我们在人为误操作或是其他意外情况下恢复数据。

举个例子，当应用升级失败，或是误操作删除了表和数据库时，我们可以通过延时节点进行数据恢复。



1.4 配置MongoDB复制

1.4.1 环境说明

系统环境说明：


```
]# cat
ease
6.9

]# uname
-r
```

高可用写得好，马克。 --—羽赐命

4. Re:Shell编程基础篇-下 @镀金的天空我刚刚也碰到mail无法发送，系统提示我没有mail这个命令，我使

```
2.6.32-696.el6.x86_64
[root@MongoDB ~]#
/etc/init.d/iptables
status
iptables: Firewall is
not running.
[root@MongoDB ~]#
getenforce
Disabled
[root@MongoDB ~]#
hostname -I
10.0.0.152 172.16.1.152
```

软件版本说明

> 本次使用的mongodb版本为：mongodb-linux-x86_64-3.2.8.tgz

1.4.2 前期准备，在root用户下操作

本次复制集复制采用Mongodb多实例进行

所有的操作都基于安装完成的mongodb服务，详情参照：http://www.cnblogs.com/clsn/p/8214194.html#_label3

```
#创建mongod用户
    useradd -u800 mongod
    echo 123456|passwd --stdin mongod
# 安装mongodb
    mkdir -p /mongodb/bin
     cd  /mongodb
     wget http://downloads.mongodb.org/linux/mongodb-linux-x86_64-rhel62-3.2.8.tgz
    tar xf  mongodb-linux-x86_64-3.2.8.tgz
    cd mongodb-linux-x86_64-3.2.8/bin/ &&\
    cp * /mongodb/bin
    chown -R mongod.mongod /mongodb
# 切换到mongod用户进行后续
```

关注一下我呗～

1.4.3 目录

用yum -y install mail x命令安装就可以使用mail了...

-- python坚持者

5. Re:Shell编程进阶篇(完结)写得很好，不过有些地方还是有点小问题哈！

-- 镀

```
for  i in 28017 28018
28019 28020
    do
      mkdir -p
/mongodb/$i/conf
      mkdir -p
/mongodb/$i/data
      mkdir -p
/mongodb/$i/log
done
```

1.4.4 配置多实例环境

编辑第一个实例配置文件

```
cat
>>/mongodb/28017/conf/mo
ngod.conf<<'EOF'
systemLog:
  destination: file
  path:
/mongodb/28017/log/mongo
db.log
  logAppend: true
storage:
  journal:
    enabled: true
  dbPath:
/mongodb/28017/data
  directoryPerDB: true
  #engine: wiredTiger
  wiredTiger:
    engineConfig:
      # cacheSizeGB: 1

directoryForIndexes:
true
    collectionConfig:
      blockCompressor:
zlib
    indexConfig:
      prefixCompression:
true
processManagement:
  fork: true
net:
  port: 28017
replication:
  oplogSizeMB: 2048
  replSetName: my_repl
EOF
```



```
28019

    \cp
/mongodb/28017/conf/mong
```

```
od.conf
/mongodb/$i/conf/
done
```

修改配置文件

```
for i in 28018 28019
28020
  do
    sed  -i
"s#28017#$i#g"
/mongodb/$i/conf/mongod.
conf
done
```

启动服务

```
for i in 28017 28018
28019 28020
  do
    mongod -f
/mongodb/$i/conf/mongod.
conf
done
```

关闭服务的方法

```
for i in 28017 28018
28019 28020
   do
     mongod --shutdown
-f
/mongodb/$i/conf/mongod.
conf
done
```

1.4.5 配置复制集

登陆数据库，配置mongodb复制

```
shell> mongo --port
28017

config = {_id:
'my_repl', members: [

{_id: 0, host:
'10.0.0.152:28017'},

{_id: 1, host:
'10.0.0.152:28018'},

{_id: 2, host:
'10.0.0.152:28019'}]
```

初始化这个配置

```
> rs.initiate(config)
```

到此复制集配置完成

1.4.6 测试主从复制

在主节点插入数据

```
my_repl:PRIMARY>
db.movies.insert([ {
"title" : "Jaws", "year"
: 1975, "imdb_rating" :
8.1 },
   { "title" : "Batman",
"year" : 1989,
"imdb_rating" : 7.6 },
  ] );
```

在主节点查看数据

```
my_repl:PRIMARY>
db.movies.find().pretty(
)
{
    "_id" :
ObjectId("5a4d9ec184b9b2
076686b0ac"),
    "title" : "Jaws",
    "year" : 1975,
    "imdb_rating" : 8.1
}
{
    "_id" :
ObjectId("5a4d9ec184b9b2
076686b0ad"),
    "title" : "Batman",
    "year" : 1989,
    "imdb_rating" : 7.6
}
```

注：在mongodb复制集当中，默认从库不允许读写。

在从库打开配置（危险）

注意：严禁在从库做任何修改操作

```
my_repl:SECONDARY>
rs.slaveOk()
my_repl:SECONDARY> show
```



```
RY>
).pretty(
{
    " id" :
```

```
ObjectId("5a4d9ec184b9b2076686b0ac"),
    "title" : "Jaws",
    "year" : 1975,
    "imdb_rating" : 8.1
}
{
    "_id" : ObjectId("5a4d9ec184b9b2076686b0ad"),
    "title" : "Batman",
    "year" : 1989,
    "imdb_rating" : 7.6
}
```

在从库查看完成在登陆到主库

1.4.7 复制集管理操作

（1）查看复制集状态：

```
rs.status();      # 查看整体复制集状态
rs.isMaster();    #  查看当前是否是主节点
```

（2）添加删除节点

```
rs.add("ip:port");      # 新增从节点
rs.addArb("ip:port"); # 新增仲裁节点
rs.remove("ip:port"); # 删除一个节点
```

注：

> *添加特殊节点时，*
> *1>可以在搭建过程中设置特殊节点*
> *2>可以通过修改配置的方式将普通从节点设置为特殊节点*
> */*找到需要改为延迟性同步的数组号*/;*

（3）配置延时节点（一般延时节点也配置成hidden）


```
priority=
slaveDela
y=120
cfg.members[2].hidden=tr
```

```
ue
```

注：这里的2是rs.conf()显示的顺序（*除主库之外*），非ID

重写复制集配置

```
rs.reconfig(cfg)
```

也可将延时节点配置为arbiter节点

```
cfg.members[2].arbiterOnly=true
```

配置成功后，通过以下命令查询配置后的属性

```
rs.conf();
```

1.4.8 副本集其他操作命令

查看副本集的配置信息

```
my_repl:PRIMARY>
rs.config()
```

查看副本集各成员的状态

```
my_repl:PRIMARY>
rs.status()
```

1.4.8.1 副本集角色切换（不要人为随便操作）

```
rs.stepDown()
rs.freeze(300)   # 锁定从，使其不会转变成主库，
freeze()和stepDown单位都是秒。
rs.slaveOk()     # 设置副本节点可读：在副本节点执行
```

插入数据

```
> use app
switched to db app
app>
db.createCollection('a')
{ "ok" : 0, "errmsg" :
                code" :
```



```
>
```

```
rs.printSlaveReplication
Info()
source:
192.168.1.22:27017
    syncedTo: Thu May 26
2016 10:28:56 GMT+0800
(CST)
    0 secs (0 hrs)
behind the primary
```

MongoDB分片（Sharding）技术

分片（sharding）是MongoDB用来将大型集合分割到不同服务器（或者说一个集群）上所采用的方法。尽管分片起源于关系型数据库分区，但MongoDB分片完全又是另一回事。

和MySQL分区方案相比，MongoDB的最大区别在于它几乎能自动完成所有事情，只要告诉MongoDB要分配数据，它就能自动维护数据在不同服务器之间的均衡。

2.1 MongoDB分片介

2.1.1 分片的目的

高数据量和吞吐量的数据库应用会对单机的性能造成较大压力,大的查询量会将单机的CPU耗尽,大的数据量对单机的存储压力较大,最终会耗尽系统的内存而将压力转移到磁盘IO上。

为了解决这些问题,有两个基本的方法: 垂直扩展和水平扩展。

垂直扩展：增加更多的CPU和存储资源来扩展容量。

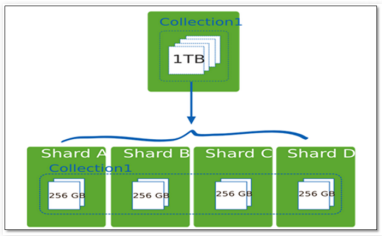
将数据集
。水平扩

2.1.2 分片设计思想

分片为应对高吞吐量与大数据量提供了方法。使用分片减少了每个分片需要处理的请求数，因此，通过水平扩展，集群可以提高自己的存储容量和吞吐量。举例来说，当插入一条数据时，应用只需要访问存储这条数据的分片.

使用分片减少了每个分片存储的数据。

例如，如果数据库1tb的数据集，并有4个分片，然后每个分片可能仅持有256 GB的数据。如果有40个分片，那么每个切分可能只有25GB的数据。



2.1.3 分片机制提供了如下三种优势

1.对集群进行抽象，让集群“不可见”

MongoDB自带了一个叫做mongos的专有路由进程。mongos就是掌握统一路口的路由器，其会将客户端发来的请求准确无误的路由到集群中的一个或者一组服务器上，同时会把接收到的响应拼装起来发回到客户端。

2.保证集群总是可读写

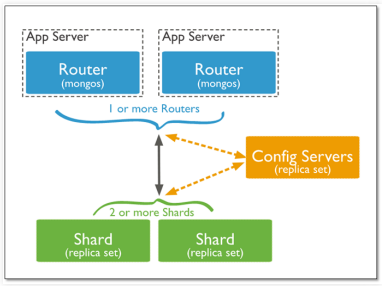
MongoDB通过多种途径来确保集群的可用性和可靠性。将MongoDB的分片和复制功能结合使用，在确保数据分片到多台服务器的同时，也确保了每分数据都有相应的备份，这样就可以确保有服务器换掉
立即接替
。

当系统需要更多的空间和

资源的时候，MongoDB便我们可以按需方便的扩充系统容量。

2.1.4 分片集群架构

组件	说明
Config Server	存储集群所有节点、分片数据路由信息。默认需要配置3个Config Server节点。
Mongos	提供对外应用访问，所有操作均通过mongos执行。一般有多个mongos节点。数据迁移和数据自动平衡。
Mongod	存储应用数据记录。一般有多个Mongod节点，达到数据分片目的。



分片集群的构造

（1）mongos ：数据路由，和客户端打交道的模块。mongos本身没有任何数据，他也不知道该怎么处理这数据，去找config server

（2）config server：所有存、取数据的方式，所有shard节点的信息，分片功能的一些配置信息，可以理解为真实数据的元数据。

（3）shard：真正的数据存储位置，以chunk为单位存数据。

Mongos本身并不持久化数据，Sharded cluster所有的元数据都会存储到Config Server，而用户的数据会议分散存储到各个shard。Mongos启动后，会从配置服务器加载元数据，开始提供服务，将用户的请求正确路由到对应的碎片。

Mongos的路由功能

当数据写入时，MongoDB Cluster根据分片键设计写入数据。

当外部语句发起数据查询时，MongoDB根据数据分布自动路由至指定节点返回数据。

2.2 集群中数据分布

2.2.1 Chunk是什么

在一个shard server内部，MongoDB还是会把数据分为chunks，每个chunk代表这个shard server内部一部分数据。chunk的产生，会有以下两个用途：

Splitting：当一个chunk的大小超过配置中的chunk size时，MongoDB的后台进程会把这个chunk切分成更小的chunk，从而避免chunk过大的情况

Balancing：在MongoDB中，balancer是一个后台进程，负责chunk的迁移，从而均衡各个shard server的负载，系统初始1个chunk，chunk size默认值64M,生产库上选择适合业务的chunk size是最好的。ongoDB会自动拆分和迁移chunks。

布（shard

（1）使用chunk来存储数据

（2）进群搭建完成之后，默认开启一个chunk，大小是64M，

（3）存储需求超过64M，chunk会进行分裂，如果单位时间存储需求很大，设置更大的chunk

（4）chunk会被自动均衡迁移。

2.2.2 chunksize的选择

适合业务的chunksize是最好的。

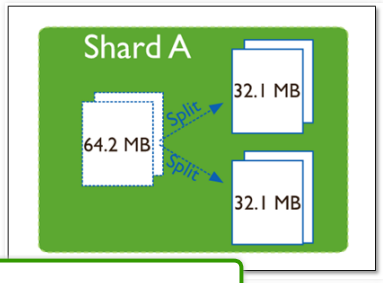
chunk的分裂和迁移非常消耗IO资源；chunk分裂的时机：在插入和更新，读数据不会分裂。

chunksize的选择：

小的chunksize：数据均衡是迁移速度快，数据分布更均匀。数据分裂频繁，路由节点消耗更多资源。大的chunksize：数据分裂少。数据块移动集中消耗IO资源。通常100-200M

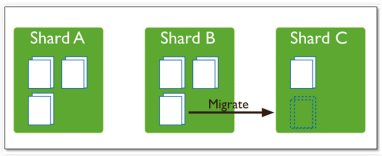
2.2.3 chunk分裂及迁移

随着数据的增长，其中的数据大小超过了配置的chunk size，默认是64M，则这个chunk就会分裂成两个。数据的增长会让chunk分裂得越来越多。



hard 上的
衡。这时
个组件
balancer 就会执行自动平衡。把chunk从chunk数量最多的

shard节点挪动到数量最少的节点。



chunkSize 对分裂及迁移的影响

MongoDB 默认的 chunkSize 为64MB，如无特殊需求，建议保持默认值；chunkSize 会直接影响到 chunk 分裂、迁移的行为。

chunkSize 越小，chunk 分裂及迁移越多，数据分布越均衡；反之，chunkSize 越大，chunk 分裂及迁移会更少，但可能导致数据分布不均。

chunkSize 太小，容易出现 jumbo chunk（即shardKey 的某个取值出现频率很高，这些文档只能放到一个 chunk 里，无法再分裂）而无法迁移；chunkSize 越大，则可能出现 chunk 内文档数太多（chunk 内文档数不能超过 250000 ）而无法迁移。

chunk 自动分裂只会在数据写入时触发，所以如果将 chunkSize 改小，系统需要一定的时间来将 chunk 分裂到指定的大小。

chunk 只会分裂，不会合并，所以即使将 chunkSize 改大，现有的 chunk 数量不会减少，但 chunk 大小会随着写入不断增长，直到达到目标大小。

2.3 数据区分

d key

据的分片是、以集合为基本单位的，集

合中的数据通过片键（Shard key）被分成多部分。其实片键就是在集合中选一个键，用该键的值作为数据拆分的依据。

所以一个好的片键对分片至关重要。片键必须是一个索引，通过sh.shardCollection加会自动创建索引（前提是此集合不存在的情况下）。一个自增的片键对写入和数据均匀分布就不是很好，因为自增的片键总会在一个分片上写入，后续达到某个阀值可能会写到别的分片。但是按照片键查询会非常高效。

随机片键对数据的均匀分布效果很好。注意尽量避免在多个分片上进行查询。在所有分片上查询，mongos会对结果进行归并排序。

对集合进行分片时，你需要选择一个片键，片键是每条记录都必须包含的，且建立了索引的单个字段或复合字段，MongoDB按照片键将数据划分到不同的数据块中，并将数据块均衡地分布到所有分片中。

为了按照片键划分数据块，MongoDB使用基于范围的分片方式或者 基于哈希的分片方式。

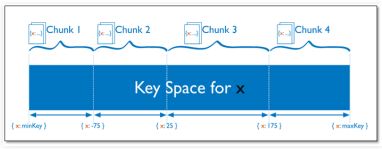
注意：

> *分片键是不可变。*
> *分片键必须有索引。*
> *分片键大小限制512bytes。*
> *分片键用于路由查询。*
> *MongoDB不接受已进行collection级分片的collection上插入无分片键的文档（也不支持*

础的分片

Sharded Cluster支持将单

Sharded Cluster支持将单个集合的数据分散存储在多shard上，用户可以指定根据集合内文档的某个字段即shard key来进行范围分片（range sharding）。

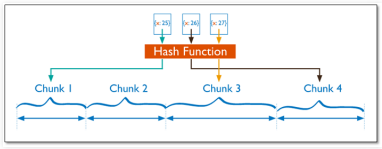


对于基于范围的分片，MongoDB按照片键的范围把数据分成不同部分。

假设有一个数字的片键:想象一个从负无穷到正无穷的直线，每一个片键的值都在直线上画了一个点。MongoDB把这条直线划分为更短的不重叠的片段，并称之为数据块，每个数据块包含了片键在一定范围内的数据。在使用片键做范围划分的系统中，拥有"相近"片键的文档很可能存储在同一个数据块中，因此也会存储在同一个分片中。

2.3.3 基于哈希的分片

分片过程中利用哈希索引作为分片的单个键，且哈希分片的片键只能使用一个字段，而基于哈希片键最大的好处就是保证数据在各个节点分布基本均匀。



对于基于哈希的分片，MongoDB计算一个字段的哈希值，并用这个哈希值来创建数据块。在使用基于哈希分片的系统中，拥有"相近"片键的文档
一个数据
离性更好

围分片互补，能将文档随机的分散到各

个chunk，充分的扩展写能力，弥补了范围分片的不足，但不能高效的服务范围查询，所有的范围查询要分发到后端所有的Shard才能找出满足条件的文档。

2.3.4 分片键选择建议

1、递增的sharding key

> *数据文件挪动小。（优势）*
> *因为数据文件递增，所以会把insert的写IO永久放在最后一片上，造成最后一片的写热点。同时，随着最后一片的数据量增大，将不断的发生迁移至之前的片上。*

2、随机的sharding key

> *数据分布均匀，insert的写IO均匀分布在多个片上。（优势）*
> *大量的随机IO，磁盘不堪重荷。*

3、混合型key

> *大方向随机递增，小范围随机分布。*
> *为了防止出现大量的chunk均衡迁移，可能造成的IO压力。我们需要设置合理分片使用策略（片键的选择、分片算法（range、hash））*

分片注意：

分片键是不可变、分片键必须有索引、分片键大小限制

用于路由查

已进行

ollection上

插入无分片键的文档（也不支

MongoDB 分片集群技术

持全值插入）

2.4 部署分片集群

本集群的部署基于1.1的复制集搭建完成。

2.4.1 环境准备

创建程序所需的目录

```
for  i in 17 18 19 20 21 22 23 24 25 26
  do
  mkdir -p /mongodb/280$i/conf
  mkdir -p /mongodb/280$i/data
  mkdir -p /mongodb/280$i/log
done
```

2.4.2 shard集群配置

编辑shard集群配置文件

```
cat > /mongodb/28021/conf/mongod.conf <<'EOF'
systemLog:
  destination: file
  path: /mongodb/28021/log/mongodb.log
  logAppend: true
storage:
  journal:
    enabled: true
  dbPath: /mongodb/28021/data
  directoryPerDB: true
  #engine: wiredTiger
  wiredTiger:
    engineConfig:
      cacheSizeGB: 1

directoryForIndexes: true
    collectionConfig:
      blockCompressor: zlib
:
pression:

.0.152
  port: 28021
replication:
```

```
  oplogSizeMB: 2048
  replSetName: sh1
sharding:
  clusterRole: shardsvr
processManagement:
  fork: true
EOF
```

复制shard集群配置文件

```
for  i in  22 23 24 25 26
  do
   \cp /mongodb/28021/conf/mongod.conf /mongodb/280$i/conf/
done
```

修改配置文件端口

```
for  i in   22 23 24 25 26
  do
    sed  -i "s#28021#280$i#g" /mongodb/280$i/conf/mongod.conf
done
```

修改配置文件复制集名称（replSetName）

```
for  i in    24 25 26
  do
    sed  -i "s#sh1#sh2#g" /mongodb/280$i/conf/mongod.conf
done
```

启动shard集群

```
for  i in  21 22 23 24 25 26
  do
    mongod -f /mongodb/280$i/conf/mongod.conf
done
```

配置复制集1


```
.0.0.152
dmin
```

```
# 配置复制集
```

```
config = {_id: 'sh1',
members: [

{_id: 0, host:
'10.0.0.152:28021'},

{_id: 1, host:
'10.0.0.152:28022'},

{_id: 2, host:
'10.0.0.152:28023',"arbi
terOnly":true}]
            }
 # 初始化配置
rs.initiate(config)
```

配置复制集2

```
mongo --host 10.0.0.152
--port 28024  admin
```

配置复制集

```
config = {_id: 'sh2',
members: [

{_id: 0, host:
'10.0.0.152:28024'},

{_id: 1, host:
'10.0.0.152:28025'},

{_id: 2, host:
'10.0.0.152:28026',"arbi
terOnly":true}]
            }
# 初始化配置
rs.initiate(config)
```

2.4.3 config集群配置

创建主节点配置文件

```
cat >
/mongodb/28018/conf/mong
od.conf <<'EOF'
systemLog:
  destination: file
  path:
/mongodb/28018/log/mongo
db.conf
  logAppend: true

storage:
  journal:
    enabled: true

  dbPath:
/mongodb/28018/data
  directoryPerDB: true
  #engine: wiredTiger
```

```
  wiredTiger:
    engineConfig:
      cacheSizeGB: 1

directoryForIndexes:
true
    collectionConfig:
      blockCompressor:
zlib
    indexConfig:
      prefixCompression:
true
net:
  bindIp: 10.0.0.152
  port: 28018
replication:
  oplogSizeMB: 2048
  replSetName:
configReplSet
sharding:
  clusterRole: configsvr
processManagement:
  fork: true
EOF
```

将配置文件分发到从节点

```
for  i in 19 20
  do
   \cp
/mongodb/28018/conf/mong
od.conf
/mongodb/280$i/conf/
done
```

修改配置文件端口信息

```
for  i in 19 20
  do
    sed  -i
"s#28018#280$i#g"
/mongodb/280$i/conf/mong
od.conf
done
```

启动config server集群

```
for  i in  18 19 20
  do
    mongod -f
/mongodb/280$i/conf/mong
od.conf
```


制集

```
.0.0.152
--port 28018  admin
```

配置复制集信息

```
config = {_id:
'configReplSet',
members: [

{_id: 0, host:
'10.0.0.152:28018'},

{_id: 1, host:
'10.0.0.152:28019'},

{_id: 2, host:
'10.0.0.152:28020'}]
           }
# 初始化配置
rs.initiate(config)
```

注：config server 使用复制集不用有arbiter节点。3.4版本以后config必须为复制集

2.4.4 mongos节点配置

修改配置文件

```
cat >
/mongodb/28017/conf/mong
os.conf <<'EOF'
systemLog:
  destination: file
  path:
/mongodb/28017/log/mongo
s.log
  logAppend: true
net:
  bindIp: 10.0.0.152
  port: 28017
sharding:
  configDB:
configReplSet/10.0.0.152
:28108,10.0.0.152:28019,
10.0.0.152:28020
processManagement:
  fork: true
EOF
```

启动mongos

```
mongos -f
/mongodb/28017/conf/mong
os.conf
```



```
10.0.0.152:28017/admin
```

添加分片节点

```
db.runCommand( {
addshard :
"sh1/10.0.0.152:28021,10
.0.0.152:28022,10.0.0.15
2:28023",name:"shard1"}
)
db.runCommand( {
addshard :
"sh2/10.0.0.152:28024,10
.0.0.152:28025,10.0.0.15
2:28026",name:"shard2"}
)
```

列出分片

```
mongos> db.runCommand( {
listshards : 1 } )
{
    "shards" : [
        {
            "_id" :
"shard2",
            "host" :
"sh2/10.0.0.152:28024,10
.0.0.152:28025"
        },
        {
            "_id" :
"shard1",
            "host" :
"sh1/10.0.0.152:28021,10
.0.0.152:28022"
        }
    ],
    "ok" : 1
}
```

整体状态查看

```
mongos> sh.status();
```

至此MongoDB的分片集群就搭建完成。

2.4.5 数据库分片配置

激活数据库分片功能

```
语法：( { enablesharding
: "数据库名称" } )
                    ommand( {
                    : "test"
```


指定分片建对集合分片，范围

片键--创建索引

```
mongos> use test
mongos>
db.vast.ensureIndex( { id: 1 } )
mongos> use admin
mongos> db.runCommand( { shardcollection : "test.vast",key : {id: 1} } )
```

集合分片验证

```
mongos> use test
mongos>
for(i=0;i<20000;i++){ db.vast1.insert({"id":i,"name":"clsn","age":70,"date":new Date()}); }
mongos> db.vast.stats()
```

插入数据的条数尽量大些，能够看出更好的效果。

2.5 分片集群的操作

2.5.1 不同分片键的配置

范围片键

```
admin>
sh.shardCollection("数据库名称.集合名称",key : {分片键: 1}  )
或
admin> db.runCommand( { shardcollection : "数据库名称.集合名称",key : {分片键: 1} } )
```

eg：

```
admin >
sh.shardCollection("test.vast",key : {id: 1}  )
或
admin> db.runCommand( { shardcollection : "test.vast",key : {id: 1} } )
```



```
sh.shardCollection( "数据库名.集合名", { 片键:
```

```
"hashed" } )
```

创建哈希索引

```
admin>
db.vast.ensureIndex( {
a: "hashed" } )
admin >
sh.shardCollection(
"test.vast", { a:
"hashed" } )
```

2.5.2 分片集群的操作

判断是否Shard集群

```
admin> db.runCommand({
isdbgrid : 1})
```

列出所有分片信息

```
admin> db.runCommand({
listshards : 1})
```

列出开启分片的数据库

```
admin> use config
config>
db.databases.find( {
"partitioned": true } )
config>
db.databases.find() //列
出所有数据库分片情况
```

查看分片的片键

```
config>
db.collections.find()
{
    "_id" : "test.vast",
    "lastmodEpoch" :
ObjectId("58a599f19c898b
bfb818b63c"),
    "lastmod" :
ISODate("1970-02-
19T17:02:47.296Z"),
    "dropped" : false,
    "key" : {
        "id" : 1
    },
    "unique" : false
}
```



```
db.printShardingStatus()
```

或

```
admin> sh.status()
```

删除分片节点

```
sh.getBalancerState()
mongos> db.runCommand( { removeShard: "shard2" } )
```

2.6 balance操作

查看mongo集群是否开启了 balance 状态

```
mongos> sh.getBalancerState()
true
```

当然你也可以通过在路由节点mongos上执行sh.status()查看balance状态。

如果balance开启，查看是否正在有数据的迁移

连接mongo集群的路由节点

```
mongos> sh.isBalancerRunning()
false
```

2.6.1 设置balance 窗口

（1）连接mongo集群的路由节点

（2）切换到配置节点

```
use config
```

（3）确定balance 开启中

```
sh.getBalancerState()
```

如果未开启，执行命令

```
sh.setBalancerState( true )
```

窗口的时间

```
ate(
ncer" },
activeWindow : { start : "<start-time>", stop : "
```

```
<stop-time>" } } },
   { upsert: true }
)
```

eg：

```
db.settings.update({ _id
: "balancer" }, { $set :
{ activeWindow : { start
: "00:00", stop : "5:00"
} } }, true )
```

当你设置了activeWindow，就不能用sh.startBalancer() 启动balance

> *NOTE*
> *The balancer window must be sufficient to complete the migration of all data inserted during the day.*
> *As data insert rates can change based on activity and usage patterns, it is important to ensure that the balancing window you select will be sufficient to support the needs of your deployment.*

（5）删除balance 窗口

```
use config
db.settings.update({ _id
: "balancer" }, { $unset
: { activeWindow : true
} })
```

2.6.2 关闭balance

默认balance 的运行可以在任何时间，只迁移需要迁移的chunk，如果要关闭balance运行，停止一段时间可以用下列方法：

ongos节点

```
lancer()
```

（3）查看balance状态

```
sh.getBalancerState()
```

（4）停止balance 后，没有迁移进程正在迁移，可以执行下列命令

```
use config
while(
sh.isBalancerRunning() )
{

print("waiting...");
          sleep(1000);
}
```

2.6.3 重新打开balance

如果你关闭了balance，准备重新打开balance

（1）连接到路由mongos节点

（2）打开balance

```
sh.setBalancerState(true
)
```

如果驱动没有命令sh.startBalancer()，可以用下列命令

```
use config
db.settings.update( {
_id: "balancer" }, {
$set : { stopped: false
} } , { upsert: true } )
```

2.6.4 关于集合的balance

关闭某个集合的balance

```
sh.disableBalancing("stu
dents.grades")
```

打开某个集合的balance

```
sh.enableBalancing("stud
ents.grades")
```


nce是开启

```
("config"
).collections.findOne({_
id :
```

```
"students.grades"}).noBa
lance;
```

2.6.5 问题解决

mongodb在做自动分片平衡的时候，或引起数据库响应的缓慢，可以通过禁用自动平衡以及设置自动平衡进行的时间来解决这一问题。

（1）禁用分片的自动平衡

```
// connect to mongos
> use config
> db.settings.update( {
_id: "balancer" }, {
$set : { stopped: true }
} , true );
```

（2）自定义 自动平衡进行的时间段

```
// connect to mongos
> use config
> db.settings.update({
_id : "balancer" }, {
$set : { activeWindow :
{ start : "21:00", stop
: "9:00" } } }, true )
```

2.7 参考文献

[1] https://www.jianshu.com/p/ddcc3643aec9

[2] https://docs.mongodb.com/manual/replication/

[3] https://docs.mongodb.com/manual/core/replica-set-architecture-three-members/

[4] http://www.mongoing.com/archives/2155

[5] https://docs.mong
ual/shar

w.ywnds.
6

[7] https://yq.aliyun.c

om/articles/32434

分类: NoSQL,其他

好文要顶 关注我 收藏该文

惨绿少年
关注 - 27
粉丝 - 296
+加关注

« 上一篇：MongoDB 入门篇
» 下一篇：MongoDB的备份与恢复

posted @ 2018-01-16 09:19 惨绿少年 阅读(5129) 评论(0) 编辑 收藏

刷新评论 刷新页面 返回顶部

注册，访问

关注一下我呗~

++源码: 大型组态工控、电力仿真CAD与