MONGODB的备份与恢复

惨绿少年 Linux运维, NoSQL, 数据库 0评论 来源：本站原创 46℃ 字体： 小 中 大 1.1

MongoDB的常用命令

```
mongoexport / mongoimport
mongodump  / mongorestore
```

有以上两组命令在备份与恢复中进行使用。

1.1.1 导出工具mongoexport

Mongodb中的mongoexport工具可以把一个collection导出成JSON格式或CSV格式的文件。可以通过参数指定导出的数据项，也可以根据指定的条件导出数据。

该命令的参数如下：

参数	参数说明
-h	指明数据库宿主机的IP
-u	指明数据库的用户名
-p	指明数据库的密码
-d	指明数据库的名字
-c	指明collection的名字
-f	指明要导出那些列
-o	指明到要导出的文件名
-q	指明导出数据的过滤条件
–type	指定文件类型
–authenticationDatabase	验证数据的名称

mongoexport备份实践

备份app库下的vast集合

```
mongoexport -h 10.0.0.152:27017 -uroot -proot --authenticationDatabase admin -d app -c vast -o /
```

注：备份文件的名字可以自定义，默认导出了JSON格式的数据。

导出CSV格式的数据

```
mongoexport -h 10.0.0.152:27017 -uroot -proot --authenticationDatabase admin  -d app -c vast --t
```

1.1.2 导入工具mongoimport

Mongodb中的mongoimport工具可以把一个特定格式文件中的内容导入到指定的collection中。该工具可以导入JSON格式数据，也可以导入CSV格式数据。

该命令的参数如下：

参数	参数说明
-h	指明数据库宿主机的IP
-u	指明数据库的用户名
-p	指明数据库的密码
-d	指明数据库的名字
-c	指明collection的名字
-f	指明要导出那些列
-o	指明到要导出的文件名
-q	指明导出数据的过滤条件
–drop	插入之前先删除原有的
–headerline	指明第一行是列名，不需要导入。
-j	同时运行的插入操作数（默认为1），并行
–authenticationDatabase	验证数据的名称

mongoimport恢复实践

将之前恢复的数据导入

```
mongoimport -h 10.0.0.152:27017 -uroot -proot --authenticationDatabase admin  -d app -c vast  --
```

将之前恢复的CSV格式数据导入

```
mongoimport -h 10.0.0.152:27017 -uroot -proot --authenticationDatabase admin -d app -c vast --ty
```

1.1.3 【实验】mysql数据迁移至mongodb数据库

mysql相关的参考文档：http://www.cnblogs.com/clsn/category/1131345.html

将mysql数据库中的mysql下的user表导出。

```
select user,host,password from mysql.user
into outfile '/tmp/user.csv'
fields terminated by ','
optionally enclosed by '"'
escaped by '"'
lines terminated by '\r\n';
```

命令说明：

```
into outfile '/tmp/user.csv'  ------导出文件位置
fields terminated by ','       ------字段间以,号分隔
optionally enclosed by '"'    ------字段用"号括起
escaped by '"'                 ------字段中使用的转义符为"
lines terminated by '\r\n';   ------行以\r\n结束
```

查看导出内容

```
[mongod@MongoDB tmp]$ cat user.csv
"root","localhost",""
"root","db02",""
"root","127.0.0.1",""
"root","::1",""
"","localhost",""
"","db02",""
"repl","10.0.0.%","*23AE809DDACAF96AF0FD78ED04B6A265E05AA257"
"mha","10.0.0.%","*F4C9AC49A736981AE2739FC2F4A1FD92B4F07929"
```

在mongodb中导入数据

```
mongoimport -h 10.0.0.152:27017 -uroot -proot --authenticationDatabase admin -d app -c user -f u
```

查看导入的内容

```
[root@MongoDB tmp]# mongo --port 27017
MongoDB shell version: 3.2.8
connecting to: 127.0.0.1:27017/test
> use app
switched to db app
> db.user.find()
{ "_id" : ObjectId("5a53206b3b42ae4683180009"), "user" : "root\tlocalhost" }
{ "_id" : ObjectId("5a53206b3b42ae468318000a"), "user" : "root\tdb02" }
{ "_id" : ObjectId("5a53206b3b42ae468318000b"), "user" : "root\t127.0.0.1" }
{ "_id" : ObjectId("5a53206b3b42ae468318000c"), "user" : "root\t::1" }
{ "_id" : ObjectId("5a53206b3b42ae468318000d"), "user" : "localhost" }
{ "_id" : ObjectId("5a53206b3b42ae468318000e"), "user" : "db02" }
{ "_id" : ObjectId("5a53206b3b42ae468318000f"), "user" : "repl\t10.0.0.%\t*23AE809DDACAF96AF0FD7
{ "_id" : ObjectId("5a53206b3b42ae4683180010"), "user" : "mha\t10.0.0.%\t*F4C9AC49A736981AE2739F
```

到此数据迁移完成。

1.2 mongodump/mongorestore实践

1.2.1 mongodump备份工具

mongodump的参数与mongoexport的参数基本一致

参数	参数说明
-h	指明数据库宿主机的IP
-u	指明数据库的用户名
-p	指明数据库的密码
-d	指明数据库的名字
-c	指明collection的名字
-o	指明到要导出的文件名
-q	指明导出数据的过滤条件
–authenticationDatabase	验证数据的名称
–gzip	备份时压缩
–oplog	use oplog for taking a point-in-time snapshot

mongodump参数实践

全库备份

```
mongodump -h 10.0.0.152:27017 -uroot -proot --authenticationDatabase admin  -o /home/mongod/back
```

备份test库

```
mongodump -h 10.0.0.152:27017 -uroot -proot --authenticationDatabase admin  -d test -o /home/mor
```

备份test库下的vast集合

```
mongodump -h 10.0.0.152:27017 -uroot -proot --authenticationDatabase admin  -d test -c vast -o /
```

压缩备份库

```
mongodump -h 10.0.0.152:27017 -uroot -proot --authenticationDatabase admin  -d test -o /home/mor
```

压缩备份单表

```
mongodump -h 10.0.0.152:27017 -uroot -proot --authenticationDatabase admin  -d test -c vast -o /
```

1.2.2 mongorestore恢复实践

mongorestore与mongoimport参数类似

参数	参数说明
-h	指明数据库宿主机的IP
-u	指明数据库的用户名
-p	指明数据库的密码
-d	指明数据库的名字
-c	指明collection的名字
-o	指明到要导出的文件名
-q	指明导出数据的过滤条件
–authenticationDatabase	验证数据的名称
–gzip	备份时压缩
–oplog	use oplog for taking a point-in-time snapshot
–drop	恢复的时候把之前的集合drop掉

全库备份中恢复单库（基于之前的全库备份）

```
mongorestore -h 10.0.0.152:27017 -uroot -proot --authenticationDatabase admin -d test --drop  /h
```

恢复test库

```
mongorestore -h 10.0.0.152:27017 -uroot -proot --authenticationDatabase admin -d test /home/mong
```

恢复test库下的vast集合

```
mongorestore -h 10.0.0.152:27017 -uroot -proot --authenticationDatabase admin -d test -c vast /h
```

–drop参数实践恢复

```
# 恢复单库
mongorestore -h 10.0.0.152:27017 -uroot -proot --authenticationDatabase admin -d test --drop /ho
# 恢复单表
mongorestore -h 10.0.0.152:27017 -uroot -proot --authenticationDatabase admin -d test -c vast --
```

1.2.3 mongoexport/mongoimport与mongodump/mongorestore的对比

1. mongoexport/mongoimport导入/导出的是JSON格式，而mongodump/mongorestore导入/导出的是BSON格式。
2. JSON可读性强但体积较大，BSON则是二进制文件，体积小但对人类几乎没有可读性。
3. 在一些mongodb版本之间，BSON格式可能会随版本不同而有所不同，所以不同版本之间用mongodump/mongorestore可能不会成功，具体要看版本之间的兼容性。当无法使用BSON进行跨版本的数据迁移的时候，使用JSON格式即mongoexport/mongoimport是一个可选项。跨版本的mongodump/mongorestore并不推荐，实在要做请先检查文档看两个版本是否兼容（大部分时候是的）。
4. JSON虽然具有较好的跨版本通用性，但其只保留了数据部分，不保留索引，账户等其他基础信息。使用时应该注意。

1.3 MongoDB中的oplog

1.3.1 什么是oplog

MongoDB 的Replication是通过一个日志来存储写操作的，这个日志就叫做oplog。

在默认情况下,oplog分配的是5%的空闲磁盘空间。通常而言,这是一种合理的设置。可以通过mongod –oplogSize来改变oplog的日志大小。

oplog是capped collection，因为oplog的特点（不能太多把磁盘填满了，固定大小）需要，MongoDB才发明了capped collection（the oplog is actually the reason capped collections were invented）. 定值大小的集合，oplogSizeMB: 2048，oplog是具有幂等性，执行过后的不会反复执行。

值得注意的是，oplog为replica set或者master/slave模式专用（standalone模式运行mongodb并不推荐）。

oplog的位置：

> oplog在local库： local。oplog
>
> master/slave 架构下：local.oplog.$main;

> replica sets 架构下：local.oplog.rs

参数说明

```
$ mongodump --help
 --oplog use oplog for taking a point-in-time snapshot
```

该参数的主要作用是在导出的同时生成一个oplog.bson文件，存放在你开始进行dump到dump结束之间所有的oplog。

oplog 官方说明https://docs.mongodb.com/manual/core/replica-set-oplog/

简单地说，在replica set中oplog是一个定容集合（capped collection），它的默认大小是磁盘空间的5%（可以通过–oplogSizeMB参数修改），位于local库的db.oplog.rs，有兴趣可以看看里面到底有些什么内容。

其中记录的是整个mongod实例一段时间内数据库的所有变更（插入/更新/删除）操作。当空间用完时新记录自动覆盖最老的记录。

所以从时间轴上看，oplog的覆盖范围大概是这样的：

其覆盖范围被称作oplog时间窗口。需要注意的是，因为oplog是一个定容集合，所以时间窗口能覆盖的范围会因为你单位时间内的更新次数不同而变化。想要查看当前的oplog时间窗口预计值.

```
sh1:PRIMARY> rs.printReplicationInfo()
configured oplog size:   2048MB   <--集合大小
```

```
log length start to end: 305451secs (84.85hrs)  <--预计窗口覆盖时间
oplog first event time:  Thu Jan 04 2018 19:39:05 GMT+0800 (CST)
oplog last event time:   Mon Jan 08 2018 08:29:56 GMT+0800 (CST)
now:                     Mon Jan 08 2018 16:33:25 GMT+0800 (CST)
```

oplog有一个非常重要的特性——**幂等性（idempotent）**。即对一个数据集合，使用oplog中记录的操作重放时，无论被重放多少次，其结果会是一样的。

举例来说，如果oplog中记录的是一个插入操作，并**不会因为你重放了两次，数据库中就得到两条相同的记录**。这是一个很重要的特性.

1.3.2 oplog.bson作用

与oplog相关的参数

参数	参数说明
–oplogReplay	重放oplog.bson中的操作内容
–oplogLimit	与–oplogReplay一起使用时，可以限制重放到的时间点

首先要明白的一个问题是数据之间互相有依赖性，比如集合A中存放了订单，集合B中存放了订单的所有明细，那么只有一个订单有完整的明细时才是正确的状态。

假设在任意一个时间点，A和B集合的数据都是完整对应并且有意义的（对非关系型数据库要做到这点并不容易，且对于MongoDB来说这样的数据结构并非合理。但此处我们假设这个条件成立），那么如果A处于时间点x，而B处于x之后的一个时间点y时，可以想象A和B中的数据极有可能不对应而失去意义。

mongodump的进行过程中并不会把数据库锁死以保证整个库冻结在一个固定的时间点，这在业务上常常是不允许的。所以就有了dump的最终结果中A集合是10点整的状态，而B集合则是10点零1分的状态这种情况。

这样的备份即使恢复回去，可以想象得到的结果恐怕意义有限。

那么上面这个oplog.bson的意义就在这里体现出来了。如果在dump数据的基础上，再重做一遍oplog中记录的所有操作，这时的数据就可以代表dump结束时那个时间点（point-in-time）的数据库状态。

1.3.3 【模拟】mongodump使用

首先我们模拟一个不断有插入操作的集合foo，

```
use clsn
for(var i = 0; i < 10000; i++) {
    db.clsn.insert({a: i});
}
```

然后在插入过程中模拟一次mongodump并指定–oplog。

```
mongodump -h 10.0.0.152 --port 28021  --oplog  -o /home/mongod/backup/oplog
```

注意：–oplog选项只对全库导出有效，所以不能指定-d选项。

因为整个实例的变更操作都会集中在local库中的oplog.rs集合中。

根据上面所说，从dump开始的时间系统将记录所有的oplog到oplog.bson中，所以我们得到这些文件：

```
[mongod@MongoDB ~]$ ll /home/mongod/backup/oplog
total 8
drwxrwxr-x 2 mongod mongod   4096 Jan  8 16:49 admin
drwxrwxr-x 2 mongod mongod   4096 Jan  8 16:49 clsn
-rw-rw-r-- 1 mongod mongod  77256 Jan  8 16:49 oplog.bson
```

查看oplog.bson中第一条和最后一条内容

```
[mongod@MongoDB oplog]$ bsondump oplog.bson  >/tmp/oplog.bson.tmp
[mongod@MongoDB oplog]$ head -1 /tmp/oplog.bson.tmp
{"ts":{"$timestamp":{"t":1515401553,"i":666}},"t":{"$numberLong":"5"},"h":{"$numberLong":"573731
[mongod@MongoDB oplog]$ tail -1 /tmp/oplog.bson.tmp
{"ts":{"$timestamp":{"t":1515401556,"i":34}},"t":{"$numberLong":"5"},"h":{"$numberLong":"-743862
```

最终dump出的数据既不是最开始的状态，也不是最后的状态，而是中间某个随机状态。这正是因为集合不断变化造成的。

使用mongorestore来恢复

```
[mongod@MongoDB oplog]$ mongorestore -h 10.0.0.152 --port 28021  --oplogReplay  --drop   /home/m
2018-01-08T16:59:18.053+0800    building a list of dbs and collections to restore from /home/mor
2018-01-08T16:59:18.066+0800    reading metadata for clsn.clsn from /home/mongod/backup/oplog/cl
2018-01-08T16:59:18.157+0800    restoring clsn.clsn from /home/mongod/backup/oplog/clsn/clsn.bsc
2018-01-08T16:59:18.178+0800    reading metadata for clsn.clsn1 from /home/mongod/backup/oplog/c
2018-01-08T16:59:18.216+0800    restoring clsn.clsn1 from /home/mongod/backup/oplog/clsn/clsn1.b
2018-01-08T16:59:18.669+0800    restoring indexes for collection clsn.clsn1 from metadata
2018-01-08T16:59:18.679+0800    finished restoring clsn.clsn1 (3165 documents)
2018-01-08T16:59:19.850+0800    restoring indexes for collection clsn.clsn from metadata
2018-01-08T16:59:19.851+0800    finished restoring clsn.clsn (10000 documents)
2018-01-08T16:59:19.851+0800    replaying oplog
2018-01-08T16:59:19.919+0800    done
```

注意黄字体，第一句表示clsn.clsn1集合中恢复了3165个文档；第二句表示重放了oplog中的所有操作。所以理论上clsn1应该有16857个文档（3165个来自clsn.bson，剩下的来自oplog.bson）。验证一下：

```
sh1:PRIMARY> db.clsn1.count()
3849
```

这就是带oplog的mongodump的真正作用。

1.3.4 从别处而来的oplog

oplog有两种来源：

> 1、mongodump时加上－oplog选项，自动生成的oplog，这种方式的oplog直接－oplogReplay 就可以恢复

> 2、从别处而来，除了－oplog之外，人为获取的oplog

例如：

```
mongodump  --port 28021 -d local -c oplog.rs
```

既然dump出的数据配合oplog就可以把数据库恢复到某个状态，那是不是拥有一份从某个时间点开始备份的dump数据，再加上从dump开始之后的oplog，如果oplog足够长，是不是就可以把数据库恢复到其后的任意状态了？**是的!**

事实上replica set正是依赖oplog的重放机制在工作。当secondary第一次加入replica set时做的initial sync就相当于是在做mongodump，此后只需要不断地同步和重放oplog.rs中的数据，就达到了secondary与primary同步的目的。

既然oplog一直都在oplog.rs中存在，我们为什么还需要在mongodump时指定–oplog呢？需要的时候从oplog.rs中拿不就完了吗？答案是肯定的，你确实可以只dump数据，不需要oplog。

在需要的时候可以再从oplog.rs中取。但前提是oplog时间窗口必须能够覆盖dump的开始时间。

及时点恢复场景模拟

模拟生产环境

```
for(i=0;i<300000;i++){ db.oplog.insert({"id":i,"name":"shenzheng","age":70,"date":new Date()});
```

插入数据的同时备份

```
mongodump -h 10.0.0.152 --port 28021  --oplog  -o /home/mongod/backup/config
```

备份完成后进行次错误的操作

```
    db.oplog.remove({});
```

备份oplog.rs文件

```
mongodump -h 10.0.0.152 --port 28021 -d local -c oplog.rs -o  /home/mongod/backup/config/oplog
```

恢复之前备份的数据

```
mongorestore -h 10.0.0.152 --port 28021--oplogReplay /home/mongod/backup/config
```

截取oplog，找到发生误删除的时间点

```
bsondump oplog.rs.bson |egrep "\"op\":\"d\"\,\"ns\":\"test\.oplog\"" |head -1
"t":1515379110,"i":1
```

复制oplog到备份目录

```
cp  /home/mongod/backup/config/oplog/oplog.rs.bson   /home/mongod/backup/config/oplog.bson
```

进行恢复，添加之前找到的误删除的点（limt）

```
mongorestore -h 10.0.0.152 --port 28021 --oplogReplay --oplogLimit "1515379110:1"  /home/mongod/
```

至此一次恢复就完成了

1.3.5 mongodb的备份准则

只针对replica或master/slave，满足这些准则MongoDB就可以进行point-in-time恢复操作：

1. 任意两次数据备份的时间间隔（第一次备份开始到第二次备份结束）不能超过oplog时间窗口覆盖范围。
2. 在上次数据备份的基础上，在oplog时间窗口没有滑出上次备份结束的时间点前进行完整的oplog备份。请充分考虑oplog备份需要的时间，权衡服务器空间情况确定oplog备份间隔。

实际应用中的注意事项：

1. 考虑到oplog时间窗口是个变化值，请关注oplog时间窗口的具体时间。
2. 在靠近oplog时间窗口滑动出有效时间之前必须要有足够的时间dump出需要的oplog.rs，请预留足够的时间，不要顶满时间窗口再备份。
3. 当灾难发生时，第一件事情就是要停止数据库的写入操作，以往oplog滑出时间窗口。特别是像上述这样的remove({})操作，瞬间就会插入大量d记录从而导致oplog迅速滑出时间窗口。

分片集群的备份注意事项

1、备份什么？

（1）configserver

（2）每一个shard节点

2、备份需要注意什么？

（1）元数据和真实数据要有对等性（blancer迁移的问题，会造成config和shard备份不一致）

> （2）不同部分备份结束时间点不一样，恢复出来的数据就是有问题的。

1.4 MongoDB监控

为什么要监控?

> 监控及时获得应用的运行状态信息，在问题出现时及时发现。

监控什么?

> CPU、内存、磁盘I/O、应用程序（MongoDB）、进程监控（ps -aux）、错误日志监控

1.4.1 MongoDB集群监控方式

```
db.serverStatus()
```

查看实例运行状态（内存使用、锁、用户连接等信息）

通过比对前后快照进行性能分析

```
"connections"     # 当前连接到本机处于活动状态的连接数
"activeClients"   # 连接到当前实例处于活动状态的客户端数量
"locks"           # 锁相关参数
"opcounters"      # 启动之后的参数
"opcountersRepl"  # 复制想关
"storageEngine"   # 查看数据库的存储引擎
"mem"             # 内存相关
```

状态:

```
db.stats()
```

显示信息说明:

```
"db" : "test" ,表示当前是针对"test"这个数据库的描述。想要查看其他数据库，可以先运行$ use datab
"collections" : 3,表示当前数据库有多少个collections.可以通过运行show collections查看当前数据库
"objects" : 13，表示当前数据库所有collection总共有多少行数据。显示的数据是一个估计值，并不是非
"avgObjSize" : 36,表示每行数据是大小，也是估计值，单位是bytes
"dataSize" : 468,表示当前数据库所有数据的总大小，不是指占有磁盘大小。单位是bytes
"storageSize" : 13312,表示当前数据库占有磁盘大小，单位是bytes,因为mongodb有预分配空间机制，为了
"numExtents" : 3,似乎没有什么真实意义。我弄明白之后再详细补充说明。
"indexes" : 1 ,表示system.indexes表数据行数。
"indexSize" : 8192,表示索引占有磁盘大小。单位是bytes
 "fileSize" : 201326592，表示当前数据库预分配的文件大小，例如test.0,test.1，不包括test.ns。
```

1.4.2 mongostat

实时数据库状态，读写、加锁、索引命中、缺页中断、读写等待队列等情况。

每秒刷新一次状态值，并能提供良好的可读性，通过这些参数可以观察到MongoDB系统整体性能情况。

```
[mongod@MongoDB oplog]$ mongostat -h 10.0.0.152 --port 28017
insert query update delete getmore command flushes mapped  vsize   res faults qr|qw ar|aw netIn
    *0    *0     *0     *0       0     1|0       0        303.0M 13.0M      0   0|0   0|0  143b
```

参数说明：

参数	参数说明
insert	每秒插入量
query	每秒查询量
update	每秒更新量
delete	每秒删除量
conn	当前连接数
qr|qw	客户端查询排队长度（读|写）最好为0，如果有堆积，数据库处理慢。
ar|aw	活跃客户端数量（读|写）
time	当前时间

mongotop命令说明：

```
[mongod@MongoDB oplog]$ mongotop  -h 127.0.0.1:27017
2018-01-08T17:32:56.623+0800    connected to: 127.0.0.1:27017

                                           ns    total    read    write    2018-01-08T17:32:
                             admin.system.roles      0ms     0ms      0ms
                             admin.system.users      0ms     0ms      0ms
                           admin.system.version      0ms     0ms      0ms
                                        app.user      0ms     0ms      0ms
            automationcore.automation.job.status      0ms     0ms      0ms
                automationcore.config.automation      0ms     0ms      0ms
       automationcore.config.automationTemplates      0ms     0ms      0ms
automationcore.config.automationTemplates_archive      0ms     0ms      0ms
        automationcore.config.automation_archive      0ms     0ms      0ms
                automationstatus.lastAgentStatus      0ms     0ms      0ms
```

mongotop重要指标

```
ns：数据库命名空间，后者结合了数据库名称和集合。
total：mongod在这个命令空间上花费的总时间。
```

```
read：在这个命令空间上mongod执行读操作花费的时间。
write：在这个命名空间上mongod进行写操作花费的时间。
```

1.4.3 db级别命令

```
db.currentOp()
```

查看数据库当前执行什么操作。

用于查看长时间运行进程。

通过（执行时长、操作、锁、等待锁时长)等条件过滤。

如果发现一个操作太长，把数据库卡死的话，可以用这个命令杀死他：> db.killOp(608605)

```
db.setProfilingLevel()
```

设置server级别慢日志

打开profiling：

> 0:不保存
>
> 1:保存慢查询日志
>
> 2:保存所有查询日志

注意:级别是对应当前的数据库，而阈值是全局的。

> 查看profiling状态
>
> 查看慢查询：system.profile
>
> 关闭profiling

企业工具ops manager官方文档： https://docs.opsmanager.mongodb.com/v3.6/

1.5 MongoDB集群性能优化方案

1.5.1 优化方向

> 硬件（内存、SSD）
>
> 收缩数据

> 增加新的机器、新的副本集
>
> 集群分片键选择
>
> chunk大小设置
>
> 预分片（预先分配存储空间）

1.5.2 存储引擎方面

WiredTiger是3.0以后的默认存储引擎，细粒度的并发控制和数据压缩提供了更高的性能和存储效率。3.0以前默认的MMAPv1也提高了性能。

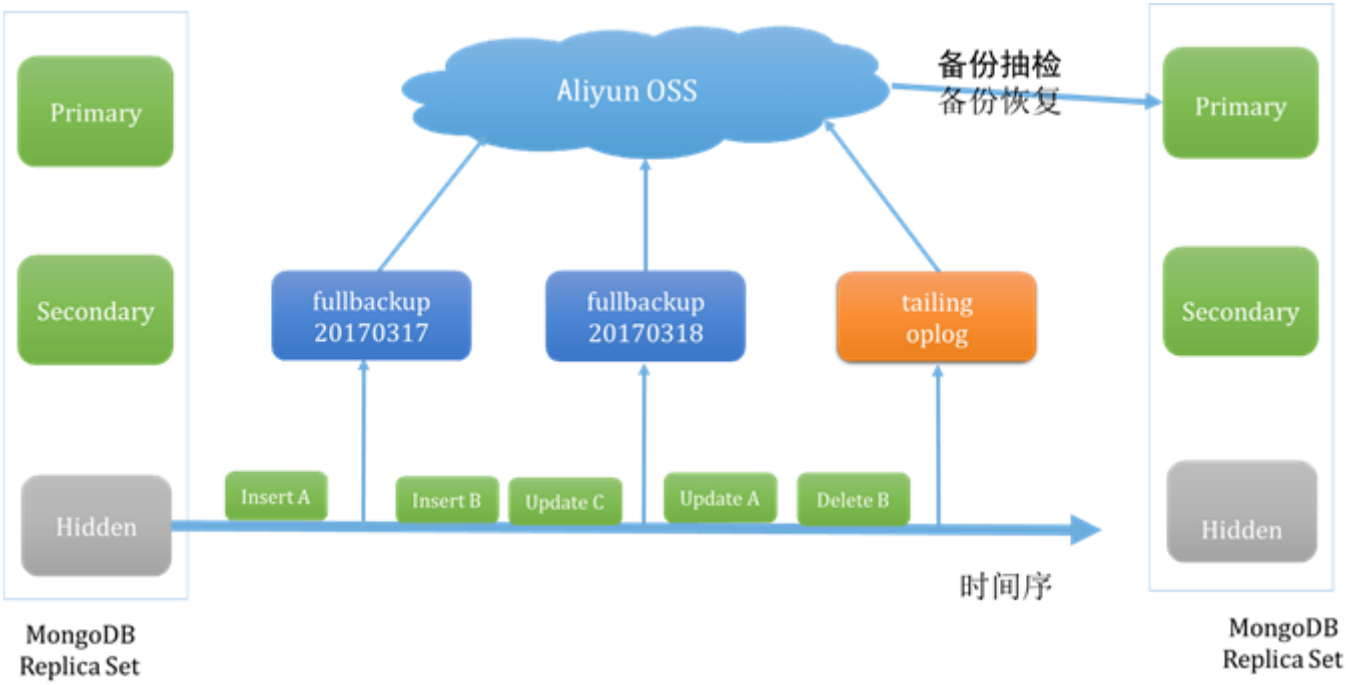
在MongoDB复制集中可以组合多钟存储引擎，各个实例实现不同的应用需求。

1.5.3 其他优化建议

> 收缩数据
>
> 预分片
>
> 增加新的机器、新的副本集
>
> 集群分片键选择
>
> chunk大小设置

1.6 附录：Aliyun 备份策略

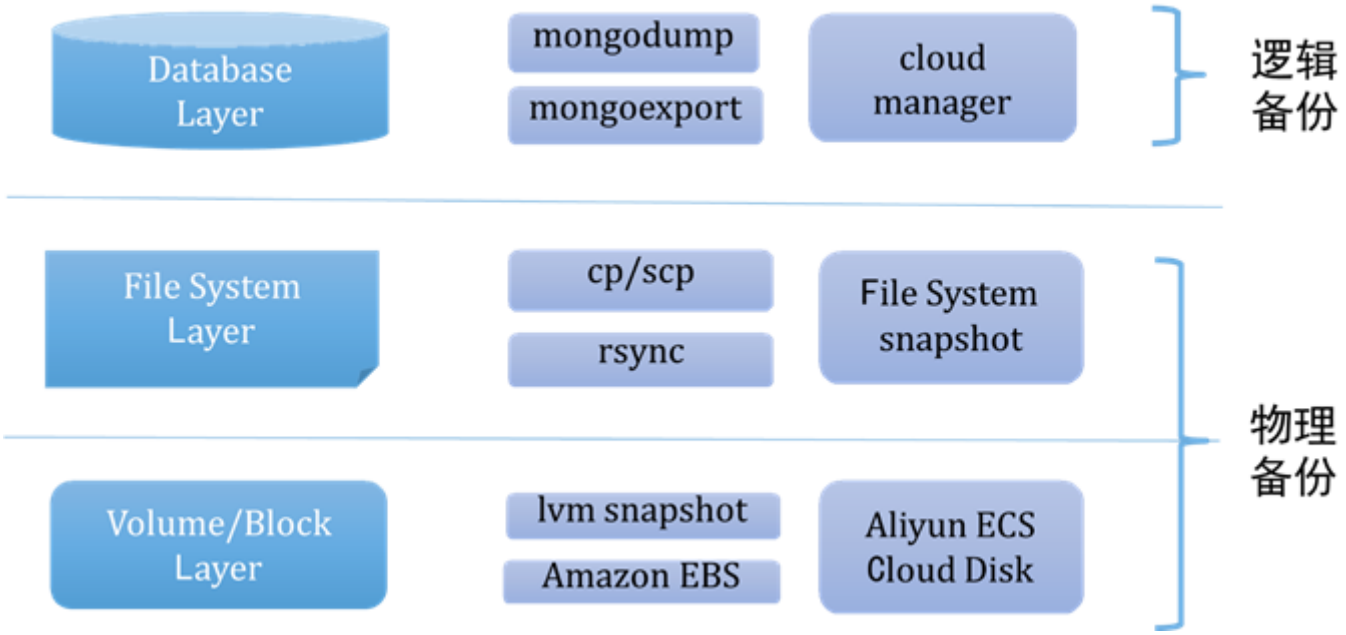
1.6.1 MongoDB云数据库备份/恢复



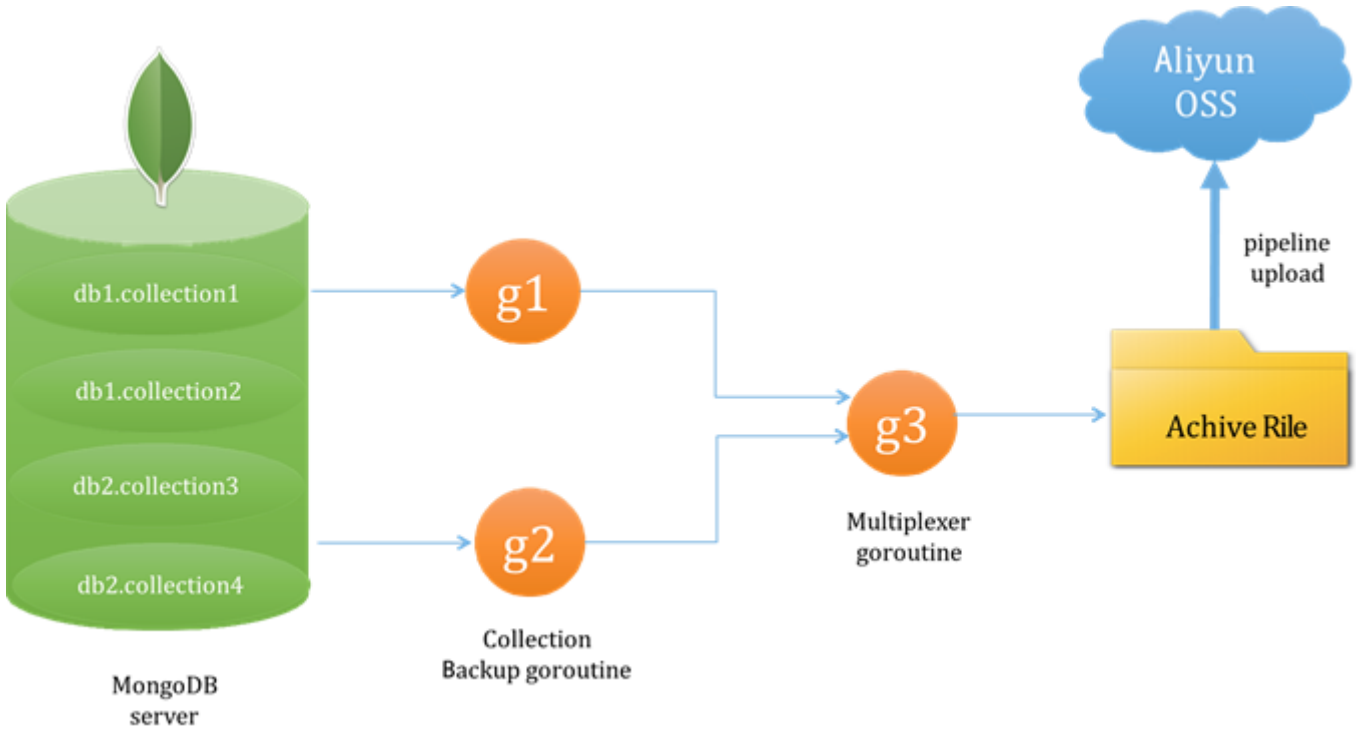
备份策略:

1. 从hidden节点备份
2. 每天一次全量备份
3. 持续拉取oplog增量备份
4. 定期巡检备份有效性
5. 恢复时克隆到新实例

1.6.2 全量备份方法



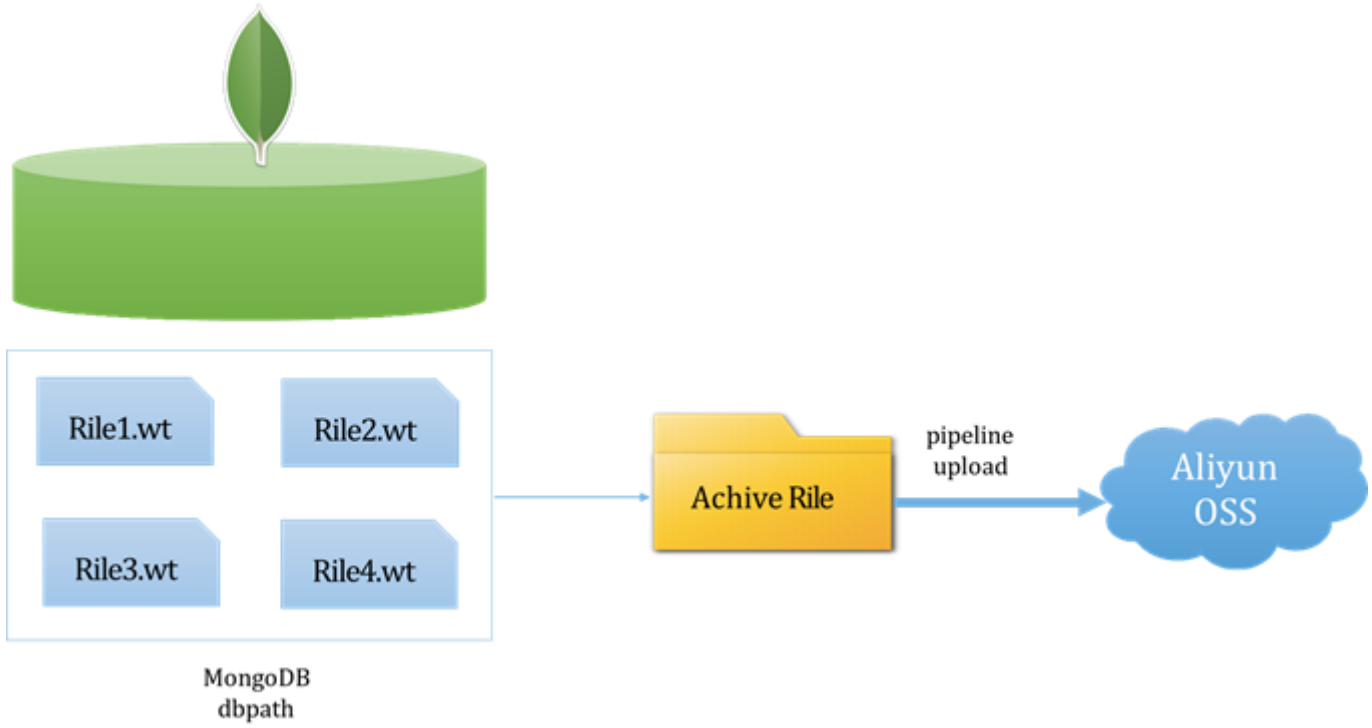
1.6.3 逻辑备份流程 – mongodump



特点:

1. 全量遍历所有数据、
2. 备份、恢复慢
3. 对业务影响较大
4. 无需备份索引、恢复时重建
5. 通用性强

1.6.4 物理备份流程



备份特点

1. 拷贝数据目录所有文件，效率高
2. 备份、恢复快
3. 对业务影响较小
4. 跟数据库版本、配置强关联

1.6.5 逻辑备份 vs 物理备份

	逻辑备份	物理备份
备份效率	**低**	**高**
	数据库接口读取数据	拷贝物理文件
恢复效率	**低**	**高**
	下载备份集 + 导入数据 + 建立索引	下载备份集 + 启动进程
备份影响	**大**	**小**
	直接与业务争抢资源	
备份集大小	比原库小	与原库相同
	无需备份索引数据	
兼容性	兼容绝大部分版本	依赖存储布局
	可跨存储引擎	

更多内容参考http://www.mongoing.com/archives/3962

1.7 参考文献

[1] http://www.cnblogs.com/yaoxing/p/mongodb-backup-rules.html

[2] http://blog.itpub.net/15498/viewspace-2073272/

[3] http://www.mongoing.com/archives/3962

[4] http://chenzhou123520.iteye.com/blog/1641319

[5] http://www.mongoing.com/oplog

[6] https://www.cnblogs.com/datazhang/p/5917861.html

赞0

如无特殊说明，文章均为本站原创，转载请注明出处

- 转载请注明来源：MongoDB的备份与恢复
- 本文永久链接地址：https://www.nmtui.com/clsn/lx247.html

该文章由 惨绿少年 发布

惨绿少年Linux www.nmtui.com