**高可用的MongoDB集群详解**

**1.序言**

MongoDB 是一个可扩展的高性能,开源,模式自由,面向文档的数据库。 它使用 C++编写。MongoDB 包含一下特点:

1.面向集合的存储:适合存储对象及JSON形式的数据。

2.动态查询:Mongo 支持丰富的查询方式,查询指令使用 JSON 形式的标记,可轻易查询文档中内嵌的对象及数组。

3.完整的索引支持:包括文档内嵌对象及数组。Mongo 的查询优化器会分析查询表达式,并生成一个高效的查询计划。

4.查询监视:Mongo包含一个监控工具用于分析数据库操作性能。

5.复制及自动故障转移:Mongo 数据库支持服务器之间的数据复制,支持主-从模式及服务器之间的相互复制。复制的主要目的是提供冗余及自动故障转移。

6.高效的传统存储方式:支持二进制数据及大型对象(如:照片或图片)。

7.自动分片以支持云级别的伸缩性:自动分片功能支持水平的数据库集群,可动态添加额外的机器。

**2.背景**

MongoDB 的主要目标是在键值对存储方式(提供了高性能和高度伸缩性) 以及传统的 RDBMS(关系性数据库)系统,集两者的优势于一身。Mongo 使用 一下场景:

1.网站数据:Mongo 非常适合实时的插入,更新与查询,并具备网站实时数据存储所需的复制及高度伸缩性。

2.缓存:由于性能很高,Mongo 也适合作为信息基础设施的缓存层。在系统重启之后,由 Mongo 搭建的持久化缓存可以避免下层的数据源过载。

3.大尺寸,低价值的数据:使用传统的关系数据库存储一些数据时可能会比较贵,在此之前,很多程序员往往会选择传统的文件进行存储。

4.高伸缩性的场景:Mongo非常适合由数十或数百台服务器组成的数据库

5.用于对象及JSON数据的存储:Mongo的BSON数据格式非常适合文档格式化的存储及查询。

注:这里需要说明下,本文旨在介绍高可用的 MongoDB 集群;这里不讨论 Hadoop 平台的 HDFS。可根据公司实际业务需求,选择合适的存储系统。

当然 MongDB 也有不适合的场景:

1.高度事务性的系统:例如银行或会计系统。传统的关系型数据库目前还是更适用于需要大量原子性复制事物的应用程序。

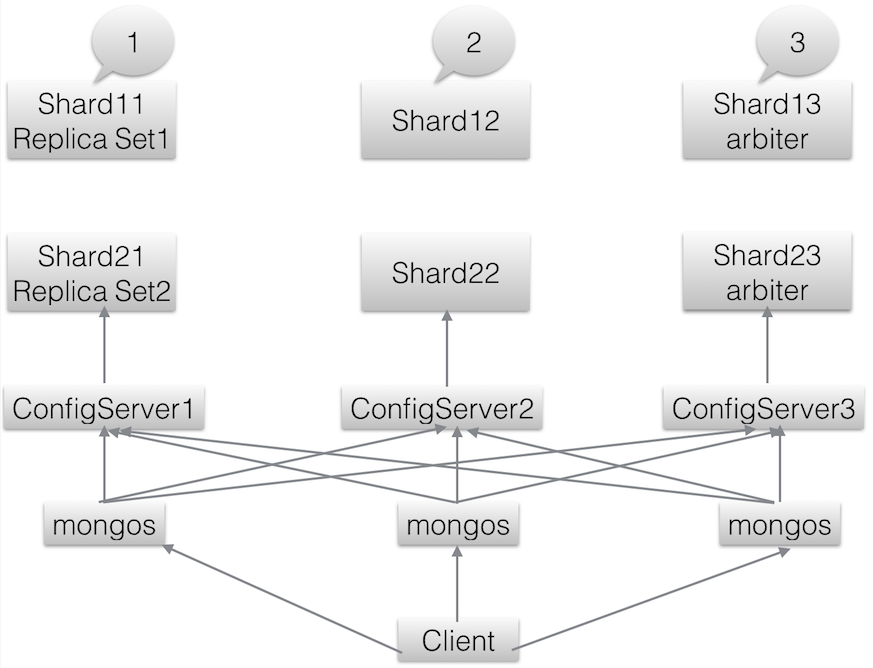
2.传统的商业智能应用:针对特定问题的 BI 数据库会对产生高度优化的查询方式。对于此类应用,数据仓库可能时更适合的选择(如Hadoop套件中的Hive)。

3.需要SQL的问题。

**3.搭建**

**3.1环境准备**

在 Mongo 的官网下载 Linux 版本安装包,然后解压到对应的目录下;由于资源有限,我们采用 Replica Sets + Sharding 方式来配置高可用。结构图如下所示:



这里我说明下这个图所表达的意思。

1.Shard服务器:使用Replica Sets确保每个数据节点都具有备份、自动容错转移、自动恢复的能力。

2.配置服务器:使用使用3个配置服务器确保元数据完整性。

3.路由进程:使用3个路由进程实现平衡,提高客户端接入性能

4.3 个分片进程:Shard11,Shard12,Shard13 组成一个副本集,提供Sharding 中 shard1 的功能。

5.3 个分片进程:Shard21,Shard22,Shard23 组成一个副本集,提供Sharding 中 Shard2 的功能。

6.3个配置服务器进程和3个路由器进程。

构建一个 mongoDB Sharding Cluster 需要三种角色:shard 服务器(ShardServer)、配置服务器(config Server)、路由进程(Route Process)

Shard 服务器shard 服务器即存储实际数据的分片,每个 shard 可以是一个 mongod 实例, 也可以是一组 mongod 实例构成的 Replica Sets.为了实现每个 Shard 内部的故障 自动转换,MongoDB 官方建议每个 shard 为一组 Replica Sets.

配置服务器

为了将一个特定的 collection 存储在多个 shard 中,需要为该 collection 指定 一个 shard key,决定该条记录属于哪个 chunk,配置服务器可以存储以下信息, 每个shard节点的配置信息,每个chunk的shard key范围,chunk在各shard 的分布情况,集群中所有 DB 和 collection 的 sharding 配置信息。

路由进程

它是一个前段路由,客户端由此接入,首先询问配置服务器需要到哪个 shard 上查询或保存记录,然后连接相应的 shard 执行操作,最后将结果返回给客户端,客 户端只需要将原本发给 mongod 的查询或更新请求原封不动地发给路由进程,而 不必关心所操作的记录存储在哪个 shard 上。

按照架构图,理论上是需要 16 台机器的,由于资源有限,用目录来替代物理机(有风险,若其中某台机器宕机,配置在该机器的服务都会 down 掉),下面给出配置表格:



**3.2.环境变量**

下面给出 MongoDB 的环境变量配置,输入命令并配置:

|  |
| --- |
| [root@mongo ~]# vi /etc/profile export MONGO\_HOME=/root/mongodb-linux-x86\_64-2.6.7  export PATH=$PATH:$MONGO\_HOME/bin |

然后保存退出，输入以下命令配置文件立即生效：

|  |
| --- |
| ?[root@mongo ~]# . /etc/profile |

**3.3.配置 Shard+Replica Sets**

我们分别启动 Shard1 的所有进程,并设置副本集为:shard1。下面给出启动 Shard1 的脚本文件。

shard11.conf

|  |
| --- |
| ?dbpath=/mongodb/data/shard11  logpath=/mongodb/log/shard11.log  pidfilepath=/mongodb/pid/shard11.pid  directoryperdb=true logappend=true  replSet=shard1  port=10011  fork=true  shardsvr=true  journal=true |

shard12.conf

|  |
| --- |
| ?dbpath=/mongodb/data/shard12  logpath=/mongodb/log/shard12.log  pidfilepath=/mongodb/pid/shard12.pid  directoryperdb=true logappend=true  replSet=shard1  port=10012  fork=true  shardsvr=true  journal=true |

shard13.conf

|  |
| --- |
| ?dbpath=/mongodb/data/shard13  logpath=/mongodb/log/shard13.log  pidfilepath=/mongodb/pid/shard13.pid  directoryperdb=true logappend=true  replSet=shard1  port=10013  fork=true  shardsvr=true  journal=true |

shard21.conf

|  |
| --- |
| dbpath=/mongodb/data/shard21 logpath=/mongodb/log/shard21.log  pidfilepath=/mongodb/pid/shard21.pid  directoryperdb=true logappend=true  replSet=shard2  port=10021  fork=true  shardsvr=true  journal=true |

shard22.conf

|  |
| --- |
| dbpath=/mongodb/data/shard22 logpath=/mongodb/log/shard22.log  pidfilepath=/mongodb/pid/shard22.pid  directoryperdb=true logappend=true  replSet=shard2  port=10022 fork=true  shardsvr=true  journal=true |

shard23.conf

|  |
| --- |
| dbpath=/mongodb/data/shard23 logpath=/mongodb/log/shard23.log  pidfilepath=/mongodb/pid/shard23.pid  directoryperdb=true logappend=true  replSet=shard2  port=10023 fork=true  shardsvr=true  journal=true |

config1.conf

|  |
| --- |
| ?dbpath=/mongodb/config/config1  logpath=/mongodb/log/config1.log  pidfilepath=/mongodb/pid/config1.pid  directoryperdb=true logappend=true  port=10031  fork=true  configsvr=true  journal=true |

config2.conf

|  |
| --- |
| dbpath=/mongodb/config/config2  logpath=/mongodb/log/config2.log  pidfilepath=/mongodb/pid/config2.pid  directoryperdb=true logappend=true  port=10032  fork=true  configsvr=true  journal=true |

config3.conf

|  |
| --- |
| dbpath=/mongodb/config/config3  logpath=/mongodb/log/config3.log  pidfilepath=/mongodb/pid/config3.pid  directoryperdb=true logappend=true  port=10033  fork=true  configsvr=true  journal=true |

route.conf

|  |
| --- |
| ?configdb=mongo:10031,mongo:10032,mongo:10033  pidfilepath=/mongodb/pid/route.pid port=10040 chunkSize=1 logpath=/mongodb/log/route.log  logappend=true fork=true |

route2.conf

|  |
| --- |
| ?configdb=mongo:10031,mongo:10032,mongo:10033  pidfilepath=/mongodb/pid/route.pid port=10042 chunkSize=1 logpath=/mongodb/log/route2.log  logappend=true fork=true |

route3.conf

|  |
| --- |
| ?configdb=mongo:10031,mongo:10032,mongo:10033  pidfilepath=/mongodb/pid/route3.pid port=10043 chunkSize=1 logpath=/mongodb/log/route3.log  logappend=true fork=true |

**3.4.启动批处理**

下面给出启动批处理的脚本，内容如下：

|  |
| --- |
| mongod -f /etc/shard11.conf mongod -f /etc/shard12.conf mongod -f /etc/shard13.conf mongod -f /etc/shard21.conf mongod -f /etc/shard22.conf mongod -f /etc/shard23.conf mongod -f /etc/config1.conf mongod -f /etc/config2.conf mongod -f /etc/config3.conf mongos -f /etc/route.conf mongos -f /etc/route2.conf mongos -f /etc/route3.conf |

**3.5.参数说明**

dbpath:数据存放目录

logpath:日志存放路径 logappend:以追加的方式记录日志

replSet:replica set 的名字

port:mongodb 进程所使用的端口号,默认为 27017 fork:以后台方式运行进程

journal:写日志

smallfiles:当提示空间不够时添加此参数

其他参数

pidfilepath:进程文件,方便停止 mongodb directoryperdb:为每一个数据库按照数据库名建立文件夹存放 bind\_ip:mongodb 所绑定的 ip 地址

oplogSize:mongodb 操作日志文件的最大大小。单位为 Mb,默认为硬盘剩余 空间的 5%

noprealloc:不预先分配存储

shardsvr:分片

configsvr:配置服务节点

configdb:配置 config 节点到 route 节点

**3.6.配置分片的表和片键**

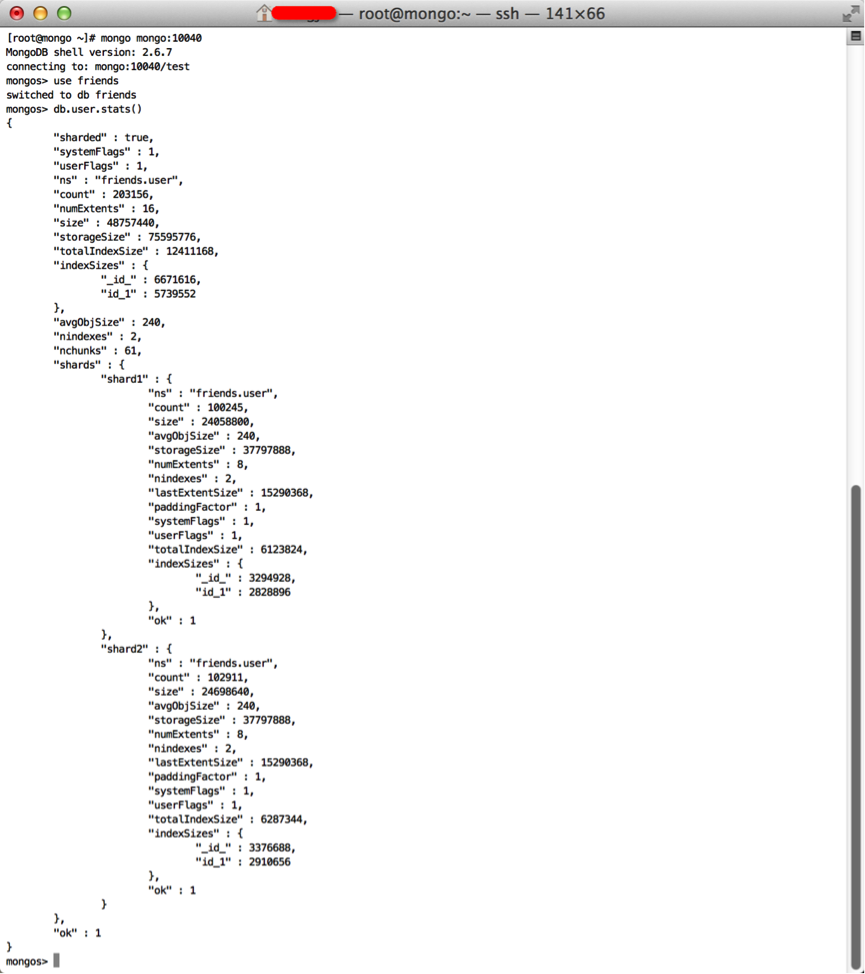
首先,我们需要登录到路由节点,这里我们登录到其中一个 10040 端口下的节点。输入如下命令:

|  |
| --- |
| ?mongo mongo:10040 use admin  db.runCommand({addshard:"shard1/mongo:10011  ,mongo:10012,mongo:10013"})  db.runCommand({addshard:"shard2/mongo:10021  ,mongo:10022,mongo:10023"})  db.runCommand({ listshards:1 }) #列出 shard  个数  db.runCommand({enablesharding:"friends"});  #创建 friends 库 db.runCommand( { shardcollection : "friends.  user",key : {id: 1},unique : true } ) #  使用 user 表来做分片,片键为 id 且唯一 |

**3.7.验证**

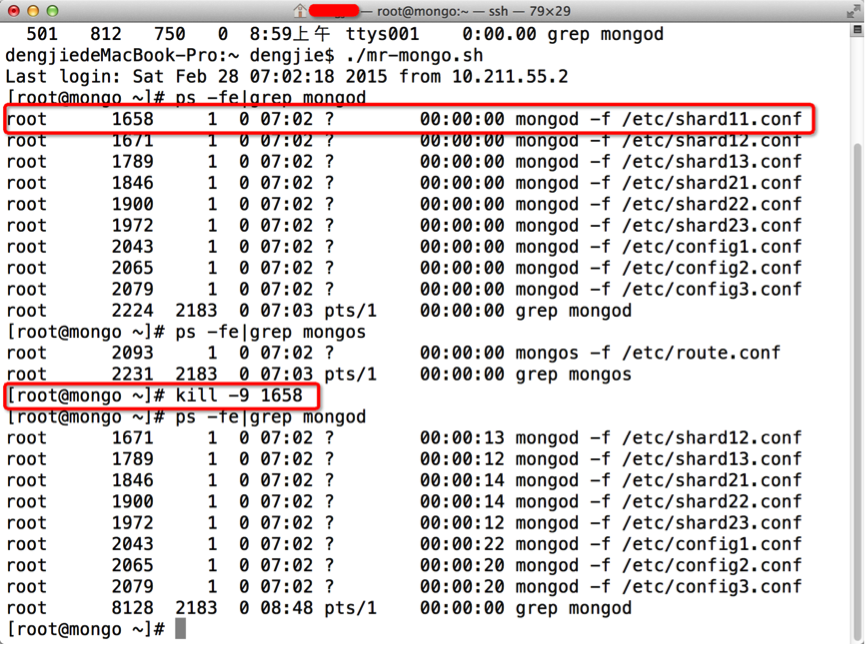
至此,整个集群的搭建完成,下面我们测试集群的高可用性。下面给出截图:

首先是查看集群的状态图 ：

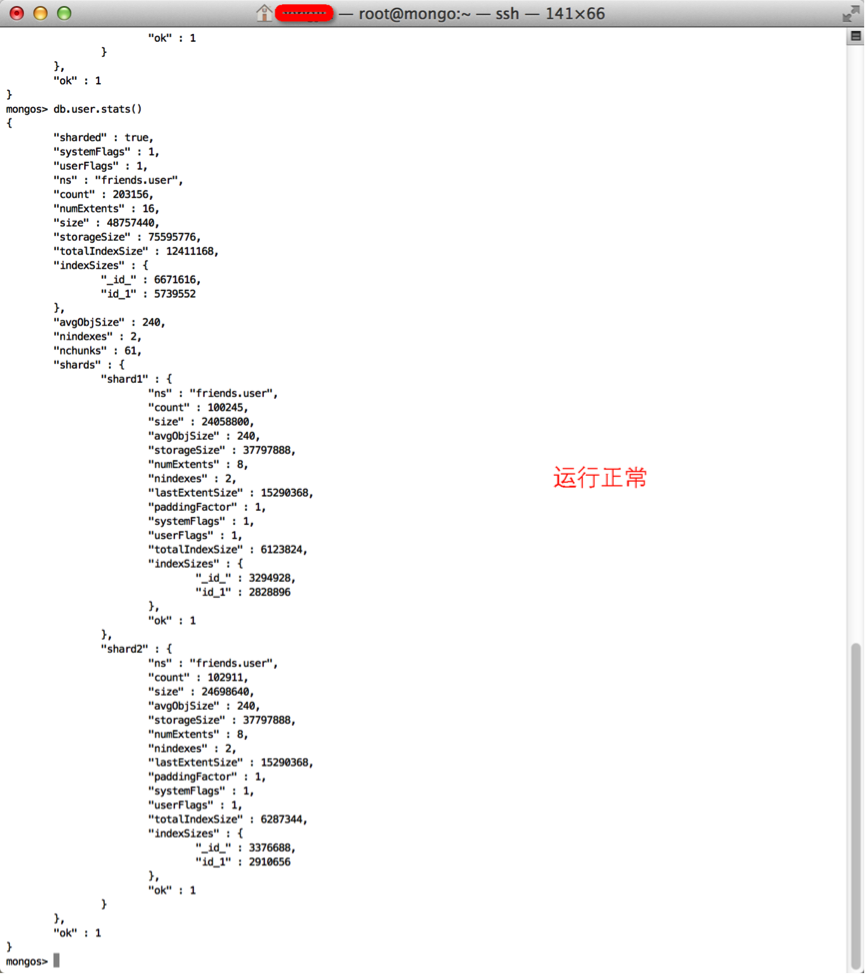


可以看到,集群中存有数据,这是我之前为了测试存的数据,注意,Mongo 只有数据达到一定量才会分片,所有我插入的数据比较大,每次测试都是 10w 的记录插入。

下面,我 kill 掉 shard11 服务,看会发生什么情况?截图如下:



这里我已经 kill 了 shard11 的进程服务。接下来,我们在 10040 端口的路由 节点输入:db.user.stats()查看状态,显示运行正常。截图如下所示:



同样可以在该路由节点插入 10w 条记录,看是否成功,下面给出插入脚本, 内容如下:

|  |
| --- |
| ?for(var i=1;i<=100000;i++)db.user.save({id:i,value1  :"1234567890",value2:"1234567890",value3:"123 4567890"  ,value4:"1234567890"}); |

**4.总结**

这片文章就分享到这里，若在研究的过程中有什么问题可以加群讨论或发送邮件给我，我会尽我所能为您解答，与君共勉！