Mongodb高级篇-性能优化

1、监控

mongodb可以通过profile来监控数据，进行优化。

查看当前是否开启profile功能用命令：`db.getProfilingLevel()`返回level等级，值为0|1|2，分别代表意思：0代表关闭，1代表记录慢命令，2代表全部。

开始profile功能为`db.setProfilingLevel(level);`

level为1的时候，慢命令默认值为100ms，更改为db.setProfilingLevel(level,slowms)如db.setProfilingLevel(1,50)这样就更改为50毫秒

通过db.system.profile.find() 查看当前的监控日志。

通过执行`db.system.profile.find({millis:{$gt:500}})`能够返回查询时间在500毫秒以上的查询命令。

这里值的含义是

ts：命令执行时间

info：命令的内容

query：代表查询

order.order： 代表查询的库与集合

reslen：返回的结果集大小，byte数

nscanned：扫描记录数量

nquery：后面是查询条件

nreturned：返回记录数及用时

millis：所花时间

如果发现时间比较长，那么就需要作优化。

比如nscanned数很大，或者接近记录总数，那么可能没有用到索引查询。

reslen很大，有可能返回没必要的字段。

nreturned很大，那么有可能查询的时候没有加限制。

mongo可以通过db.serverStatus()查看mongod的运行状态

2、索引

如果发现查询时间相对长，那么就需要做优化。首选就是为待查询的字段建立索引，不过需要特别注意的是，索引不是万能灵药。如果需要查询超过一半的集合数据，索引还不如直接遍历来的好。
索引的原理是通过建立指定字段的B树，通过搜索B树来查找对应document的地址。这也就解释了如果需要查询超过一半的集合数据，直接遍历省去了搜索B树的过程，效率反而会高。
关于索引，索引列颗粒越小越好，什么叫颗粒越小越好？在索引列中每个数据的重复数量称为颗粒，也叫作索引的基数。如果数据的颗粒过大，索引就无法发挥该有的性能。例如，我们拥有一个"age"列索引，如果在"age"列中，20岁占了50%，如果现在要查询一个20岁，名叫"Tom"的人，我们则需要在表的50%的数据中查询，索引的作用大大降低。所以，我们在建立索引时要尽量将数据颗粒小的列放在索引左侧，以保证索引发挥最大的作用。

3、exlpain查询执行情况

执行命令：

```
> db.order.find({ "status": 1.0, "user.uid": { $gt: 2663199.0 }
}).explain()
{
    "cursor" : "BasicCursor",#游标类型
    "nscanned" : 2010000,#扫描数量
    "nscannedObjects" : 2010000,#扫描对象
    "n" : 337800,#返回数据
    "millis" : 2838,#耗时
    "nYields" : 0,
    "nChunkSkips" : 0,
    "isMultiKey" : false,
    "indexOnly" : false,
    "indexBounds" : {#使用索引（这里没有）

    }
}
```

通过这些信息就能判断查询时如何执行的了

4、数据库设计优化

在项目设计阶段，明确集合的用途是对性能调优非常重要的一步。
从性能优化的角度来看，集合的设计我们需要考虑的是集合中数据的常用操作，例如我们需要设计一个日志（log）集合，日志的查看频率不高，但写入频率却

很高，那么我们就可以得到这个集合中常用的操作是更新（增删改）。如果我们要保存的是城市列表呢？显而易见，这个集合是一个查看频率很高，但写入频率很低的集合，那么常用的操作就是查询。

对于频繁更新和频繁查询的集合，我们最需要关注的重点是他们的范式化程度，假设现在我们需要存储一篇图书及其作者，在MongoDB中的关联就可以体现为以下几种形式：

1.完全分离（范式化设计）

示例1：

```
View Code
{
     "_id" : ObjectId("5124b5d86041c7dca81917"),
     "title" : "如何使用MongoDB",
      "author" : [
                ObjectId("144b5d83041c7dca84416"),
               ObjectId("144b5d83041c7dca84418"),
               ObjectId("144b5d83041c7dca84420"),
     ]
 }
```

我们将作者(comment) 的id数组作为一个字段添加到了图书中去。这样的设计方式是在非关系型数据库中常用的，也就是我们所说的范式化设计。在MongoDB中我们将与主键没有直接关系的图书单独提取到另一个集合，用存储主键的方式进行关联查询。当我们要查询文章和评论时需要先查询到所需的文章，再从文章中获取评论id，最后用获得的完整的文章及其评论。在这种情况下查询性能显然是不理想的。但当某位作者的信息需要修改时，范式化的维护优势就凸显出来了，我们无需考虑此作者关联的图书，直接进行修改此作者的字段即可。

2.完全内嵌（反范式化设计）

示例2：

```
View Code
{
       "_id" : ObjectId("5124b5d86041c7dca81917"),
       "title" : "如何使用MongoDB",
       "author" : [
                 {
                           "name" : "丁磊"
                           "age" : 40,
                           "nationality" : "china",
                 },
```

```
                {
                        "name" : "马云"
                        "age" : 49,
                        "nationality" : "china",
                },
                {
                        "name" : "张召忠"
                        "age" : 59,
                        "nationality" : "china",
                },
        ]
  }
```

在这个示例中我们将作者的字段完全嵌入到了图书中去，在查询的时候直接查询图书即可获得所对应作者的全部信息，但因一个作者可能有多本著作，当修改某位作者的信息时时，我们需要遍历所有图书以找到该作者，将其修改。

3.部分内嵌（折中方案）

示例3：

```
View Code
{
       "_id" : ObjectId("5124b5d86041c7dca81917"),
       "title" : "如何使用MongoDB",
       "author" : [
               {
                        "_id" : ObjectId("144b5d83041c7dca84416"),
                        "name" : "丁磊"
                },
                {
                        "_id" : ObjectId("144b5d83041c7dca84418"),
                        "name" : "马云"
                },
                {
                        "_id" : ObjectId("144b5d83041c7dca84420"),
                        "name" : "张召忠"
                },
        ]
  }
```

这次我们将作者字段中的最常用的一部分提取出来。当我们只需要获得图书和作者名时，无需再次进入作者集合进行查询，仅在图书集合查询即可获得。

这种方式是一种相对折中的方式，既保证了查询效率，也保证的更新效率。但这样的方式显然要比前两种较难以掌握，难点在于需要与实际业务进行结合来寻找合适的提取字段。如同示例3所述，名字显然不是一个经常修改的字段，这样的

字段如果提取出来是没问题的，但如果提取出来的字段是一个经常修改的字段（比如age）的话，我们依旧在更新这个字段时需要大范围的寻找并依此进行更新。
在上面三个示例中，第一个示例的更新效率是最高的，但查询效率是最低的，而第二个示例的查询效率最高，但更新效率最低。所以在实际的工作中我们需要根据自己实际的需要来设计表中的字段，以获得最高的效率。

5、其他方法

热数据法
可能你的数据集非常大，但是这并不那么重要，重要的是你的热数据集有多大，你经常访问的数据有多大(包括经常访问的数据和所有索引数据)。使用MongoDB，你最好保证你的热数据在你机器的内存大小之下，保证内存能容纳所有热数据。
文件系统法
MongoDB的数据文件是采用的预分配模式，并且在Replication里面，Master和Replica Sets的非Arbiter节点都是会预先创建足够的空文件用以存储操作日志。这些文件分配操作在一些文件系统上可能会非常慢，导致进程被Block。所以我们应该选择那些空间分配快速的文件系统。这里的结论是尽量不要用ext3，用ext4或者xfs。
硬件法
这里的选择包括了对磁盘RAID的选择，也包括了磁盘与SSD的对比选择。

其他

如果数据文件大于系统内存，查询速度会下降几个数量级，因为mongodb是内存数据库。我以前测试过，1000万数据的时候没有索引情况下查询可能会几秒钟甚至更久。
这种情况，你最好给经常查询的项创建索引，有索引以后查询速度会非常非常非常的快。
另外一点是数据索引如果大于内存，速度也会下降很多。而且对于多条件查询，如果你查询的顺学和索引顺序不同，也不能使用索引。这个要慢慢摸索
如果你使用了replica set，这个会影响写入速度的，三个replica set，速度会降低到三分之一。