MongoDB优化与一些需要注意的细节

这里总结下这段时间使用mongo的心得，列出了几个需要注意的地方。

1. 系统参数及mongo参数设置

mongo参数主要是storageEngine和directoryperdb，这两个参数一开始不选定后续就无法再更改。
directoryperdb主要是将数据库分文件夹存放，方便后续的备份及数据迁移。
storageEngine（存储引擎）默认使用的是MMAPv1，推荐使用3.0新加入的引擎wiredTiger。经实际使用wiredTiger占用的磁盘空间是MMAP的1/5，索引大小是其1/2，查询速度也提高很多，更重要的是该引擎提供了document级别的锁，当集合插入或更新数据时不需要阻塞读操作了。唯一的问题是市面上支持该引擎查询的工具不多，MongoVUE无法查到该引擎存储的集合，NosqlManager-mongo可以查到但需要.net环境支持。个人觉得熟悉下mongo command用mongo shell就足够了，所以还是强烈推荐使用wiredTiger引擎。

1. 无需对集合进行水平切分

由于之前一直使用关系型数据库，关系型数据库当单表数据量超大时经常使用的一直方法是对数据表进行分表。在使用mongo时便很自然的觉得这招仍然有用。由于该系统的分表都是动态生成的，做到后面发现这招对mongo带来的性能提升远远抵不过维护成本的增加。
分析一下关系型数据库分表会提高性能的最大原因是很多关系型数据库一张表是一个文件，分表可以避免一个文件过大所造成数据提取速度变慢。但是mongo并不是这样存储的，所以这条并不成立了。
用过的都知道mongo对索引的依赖非常大，如果集合不能一开始就设计好，那后续索引就得写脚本来创建。
这里贡献个给mongo大表动态创建索引的脚本：

```
//找出所有数据量超过100w的信息集合info，并为其创建索引
eval(function () {
var infos = [];
var collNames = db.getCollectionNames();
for (var i = 0; i < collNames.length; i++) {
var collName = collNames[i];
var collSize = db.getCollection(collName).count();
if (collSize > 1000000 && collName.indexOf(“info_”)==0) {
db.getCollection(collName).ensureIndex({publishDate:-1,blendedScore:-1,publishTime:-1,isRubbish:1},
{name:”ScoreSortIdx”,background:true});
db.getCollection(collName).ensureIndex({similarNum:-1,publishTime:-1,isRubbish:1},
{name:”HotSortIdx”,background:true});
db.getCollection(collName).ensureIndex({publishTime:-1,isRubbish:1},
{name:”TimeSortIdx”,background:true});
infos.push(“name:” + collName + “索引创建成功”);
}
}
return infos;
}());
```

这么看动态创建索引勉强还是可以解决的，但是最坑的一个地方是sharding完全没办法做了。shard需要指定要shard的集合和分区键，这个就没法提前动态指定了。所以mongo集合不需要做水平切分（至少千万级不需要了，更大直接shard掉），只需要按业务分开就可以了。

1. 使用Capped Collection

有人使用mongo做数据缓存，而且是缓存固定数量的数据，仍然用正常的集合，然后定期清理数据。其实这时用capped collection性能会好很多。

1. 生产环境一定要用副本集

很多人线上环境还是用单机版，虽然部署快但是很多mongo自然提供的功能都没有用到像自动故障转移、读写分离，这些对后续系统扩容及性能优化太重要了。我想会使用mongo的应该是数据量达到一定级别，查询性能会非常重要，所以强烈建议上线时直接使用副本集。

1. 学会使用explain

之前一直习惯用工具来查询，现在发现应该多使用mongo shell命令来查询，并使用explain查看查询计划。另外在寻找最优索引的时候hint命令也是非常有用的。

1

```
db.info.find({publishDate:{$gte:20160310,$lte:20160320},isRubbish:{$in:[0,1]},title:
{$regex:".*test.*"},$or:[{useId:10},{groupId:20}]}).explain(“executionStats”);
```

6. 写操作频繁无法使用读写分离

由于系统写操作较多，造成各种w级别锁经常出现（这种锁一般是block read的）而且系统对于数据一致性要求不会太多（大多是后台写入，前台读取，因此允许有一定延迟）所以想用副本集来做读写分离。当真正测试后发现副本集上的读取也经常出现阻塞的情况。通过db.currentOp()发现经常出现一个op:none的操作在申请global write lock，这时所有操作的状态都是在waitingForLock:true，这个问题google了很久都没找到解决方法。后面在官方文档有关并发的FAQ中发现下面这个大坑：

How does concurrency affect secondaries?
In replication, MongoDB does not apply writes serially to secondaries.
Secondaries collect oplog entries in batches and then apply those
batches in parallel. Secondaries do not allow reads while applying the
write operations, and apply write operations in the order that they
appear in the oplog.

原来mongodb的副本在复制主节点数据执行oplog的时候，读取是被阻塞的，这基本宣告无法在副本上去读取数据了，白白耗费了几天精力。所以mongo官方不推荐做读写分离，原来坑是在这里。。。其实写多读少的情况做读写分离作用也不大，因为性能瓶颈主要是在写入，读取一般不消耗多少资源（另外wiredTiger引擎的锁做到了doc级别，所以锁的情况相对较少）。官方推荐的做法是shard，可以有效的将写入分配到多台服务器提高写入速度，使系统实现水平扩容。

7、千万不要让磁盘满了

80%的时候就要开始注意从集拆分片，如果你的数据增长特别快，很可能你还没有拆分磁盘就满了导致MongoDB挂掉了。如果数据量很大，尽量使用分片，不要使用副本集，做好磁盘容量规划，就是使用分片了也提前扩容，毕竟chunk迁移还是那么的慢。

8、安全风险

MongoDB是默认不提示用户设置密码的，所以，如果你没有配置密码又把MongoDB放在公网上面了，那么「恭喜」，你可能已经成为了肉鸡

9、数据库级锁

MongoDB的锁机制和一般关系数据库如 MySQL（InnoDB），Oracle 有很大的差异，InnoDB 和 Oracle 能提供行级粒度锁，而 MongoDB 只能提供 库级粒度锁，这意味着当 MongoDB 一个写锁处于占用状态时，其它的读写操作都得干等。

初看起来库级锁在大并发环境下有严重的问题，但是 MongoDB 依然能够保持大并发量和高性能，这是因为 MongoDB 的锁粒度虽然很粗放，但是在锁处理机制和关系数据库锁有很大差异，主要表现在：

MongoDB 没有完整事务支持，操作原子性只到单个 document 级别，所以通常操作粒度比较小；
MongoDB 锁实际占用时间是内存数据计算和变更时间，通常很快；

MongoDB 锁有一种临时放弃机制，当出现需要等待慢速 IO 读写数据时，可以先临时放弃，等 IO 完成之后再重新获取锁。

通常不出问题不等于没有问题，如果数据操作不当，依然会导致长时间占用写锁，比如下面提到的前台建索引操作，当出现这种情况的时候，整个数据库就处于完全阻塞状态，无法进行任何读写操作，情况十分严重。

解决问题的方法，尽量避免长时间占用写锁操作，如果有一些集合操作实在难以避免，可以考虑把这个集合放到一个单独的 MongoDB 库里，因为 MongoDB 不同库锁是相互隔离的，分离集合可以避免某一个集合操作引发全局阻塞问题。