Mongodb性能调优 -性能优化建议

摘要

1. MongoDB 适用场景简介
2. Mongodb 性能监控与分析
3. Mongodb 性能优化建议

关于Mongodb的几个大事件

1.根据美国数据库知识大全官网发布的DB热度排行，Mongodb的热度排名从2014年的第5名，在2015年跃升为第4名，仅次于主流DB（Oracle、MySQL、SQLServer）之后。

282 systems in ranking, September 2015

Rank			DBMS	Database Model	Score		
Sep 2015	Aug 2015	Sep 2014			Sep 2015	Aug 2015	Sep 2014
1.	1.	1.	Oracle	Relational DBMS	1463.37	+10.35	-3.53
2.	2.	2.	MySQL	Relational DBMS	1277.75	-14.28	-19.39
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1097.83	-10.83	-111.04
4.	4.	↑ 5.	MongoDB ⊞	Document store	300.57	+5.91	+59.58
5.	5.	↓ 4.	PostgreSQL	Relational DBMS	286.18	+4.31	+30.38
6.	6.	6.	DB2	Relational DBMS	209.14	+7.91	+12.11
7.	7.	7.	Microsoft Access	Relational DBMS	146.00	+1.79	+5.52
8.	8.	↑ 9.	Cassandra ⊞	Wide column store	127.60	+13.60	+39.74
9.	9.	↓ 8.	SQLite	Relational DBMS	107.66	+1.84	+15.04
10.	10.	↑ 12.	Redis ⊞	Key-value store	100.65	+1.85	+26.05

2.2015第六届中国数据库技术大会(DTCC)上，Mongodb高调宣布收购开源引擎WiredTiger，性能在3.0版本上实现了7~10倍的提升。

Mongodb 适用场景简介

适用场景

1. 实时的CRU操作，如网站、论坛等实时数据存储
2. 高伸缩性，可以分布式集群，动态增删节点
3. 存储大尺寸、低价值数据
4. 缓存
5. BSON结构对象存储

不适用场景

1. 高度事务性操作，如银行或会计系统
2. 传统商业智能应用，如提供高度优化的查询方式
3. 需要SQL的问题
4. 重要数据，关系型数据

Mongodb 性能监控与分析

mongostat

1. faults/s：每秒访问失败数，即数据被交换出物理内存，放到SWAP。
若过高（一般超过100），则意味着内存不足。
vmstat & iostat & iotop

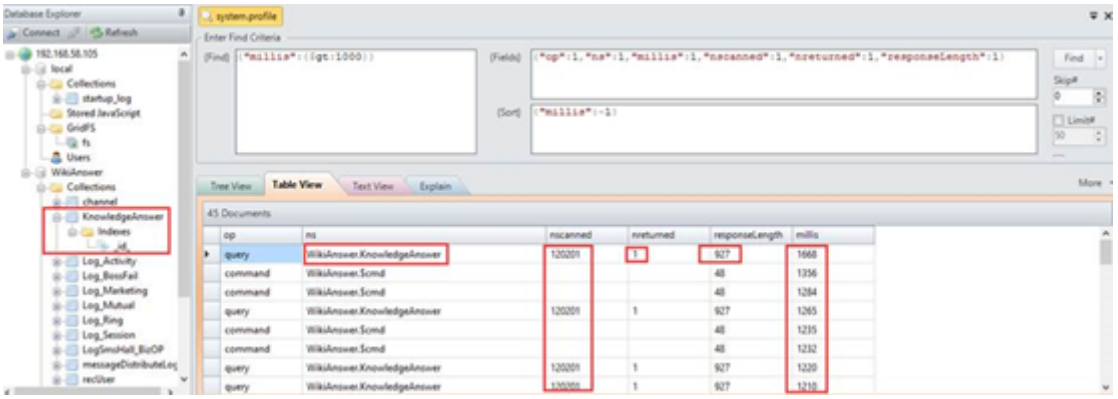
```
[root@i-9AAF38D3 ~]# vmstat
procs -----------memory---------- ---swap-- -----io---- -
 r  b   swpd   free   buff  cache   si   so    bi    bo
 0  0      0 2261504  16260 560748    0    0     0     3
```

si：每秒从磁盘读入虚拟内存的大小，若大于0，表示物理内存不足。
so：每秒虚拟内存写入磁盘的大小，若大于0，同上。
mongostat
2. idx miss %：BTree 树未命中的比例，即索引不命中所占百分比。
若过高，则意味着索引建立或使用不合理。
db.serverStatus()
indexCounters” : {
“btree” : {
“accesses” : 2821726, #索引被访问数
“hits” : 2821725, #索引命中数
“misses” : 1, #索引偏差数
“resets” : 0, #复位数
“missRatio” : 3.543930204420982e-7 #未命中率
}
mongostat
3. locked %：全局写入锁占用了机器多少时间。当发生全局写入锁时，所有查询操作都将等待，直到写入锁解除。
若过高（一般超过50%），则意味着程序存在问题。
db.currentOp()
{
“inprog” : [ ],
“fsyncLock” : 1, #为1表示MongoDB的fsync进程（负责将写入改变同步到磁盘）不允许其他进程执行写数据操作
“info” : “use db.fsyncUnlock() to terminate the fsync write/snapshot lock”
}
mongostat

4. q r|w： 等待处理的查询请求队列大小。
若过高，则意味着查询会过慢。
db.serverStatus()
“currentQueue” : {
“total” : 1024, #当前需要执行的队列
“readers” : 256, #读队列
“writers” : 768 #写队列
}
mongostat
5. conn： 当前连接数。
高并发下，若连接数上不去，则意味着Linux系统内核需要调优。
db.serverStatus()
“connections” : {
“current” : 3, #当前连接数
“available” : 19997 #可用连接数
}
6.连接数使用内存过大

```
shell> cat /proc/$(pidof mongod)/limits | grep stack | awk -F 'size' '{print int($NF)/1024}'
```

将连接数使用Linux栈内存设小，默认为10MB（10240）
shell> ulimit -s 1024
优化器Profile
db.setProfilingLevel(2);
0 – 不开启
1 – 记录慢命令 (默认为>100ms)
2 – 记录所有命令
info: #本命令的详细信息
reslen: #返回结果集的大小
nscanned: #本次查询扫描的记录数
nreturned: #本次查询实际返回的结果集
millis: #该命令执行耗时(毫秒)



1. 表KnowledgeAnswer未建立有效索引（建议考虑使用组合索引）
2. 存在大量慢查询，均为表KnowledgeAnswer读操作，且响应超过1秒
3. 每次读操作均为全表扫描，意味着耗用CPU(25% * 8核)
4. 每次返回的记录字节数近1KB，建议过滤不必要的字段，提高传输效率

执行计划Explain

db.test.find({age: “20” }).hint({age:1 }).explain();

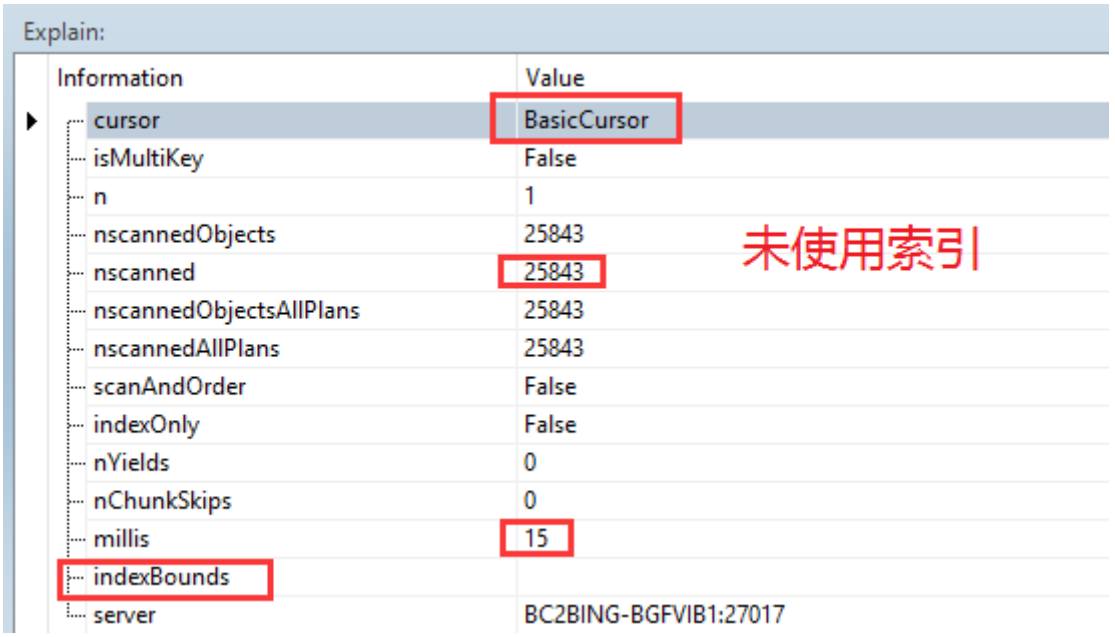
cursor: 返回游标类型(BasicCursor 或 BtreeCursor)

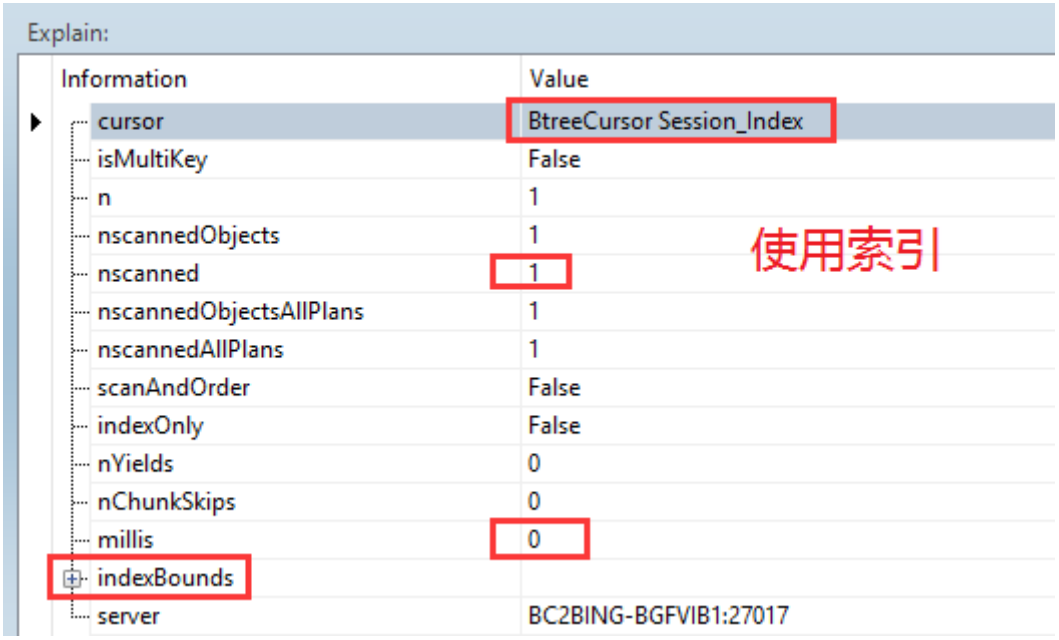
nscanned: 被扫描的文档数量

n: 返回的文档数量

millis: 耗时(毫秒)

indexBounds: 所使用的索引





1. 在查询条件、排序条件、统计条件的字段上选择创建索引

db.student.ensureIndex({name:1,age:1} , {backgroud:true});

注意：

□ 最新或最近记录查询，结合业务需要正确使用索引方向：逆序或顺序

□ 建议索引建立操作置于后台运行，降低影响

□ 实际应用过程中多考虑使用复合索引

□ 使用limit()限定返回结果集的大小，减少数据库服务器的资源消耗，以及网络传输的数据量

db.posts.find().sort({ts:-1}).limit(10);

2. 只查询使用到的字段，而不查询所有字段

db.posts.find({ts:1,title:1,author:1,abstract:1}).sort({ts:-1}).limit(10);

3. 基于Mongodb分布式集群做数据分析时，MapReduce性能优于count、distinct、group等聚合函数

SQL 查询	MapReduce 函数
SELECT	db.sales.runCommand({ mapreduce : "sales" ,

ProductCategory,
ProductSubCategory,
ProductName,
	map:function()
	{
	emit(
	// Key
	{key0:this.ProductCategory,
	key1:this.ProductSubCategory,
	key2:this.ProductName},
	// Values
	this.SalesAmt);
	},

SUM(SalesAmt)	reduce:function(key,values)
	{
	var result = Sum(key, values);
	return result;
	}

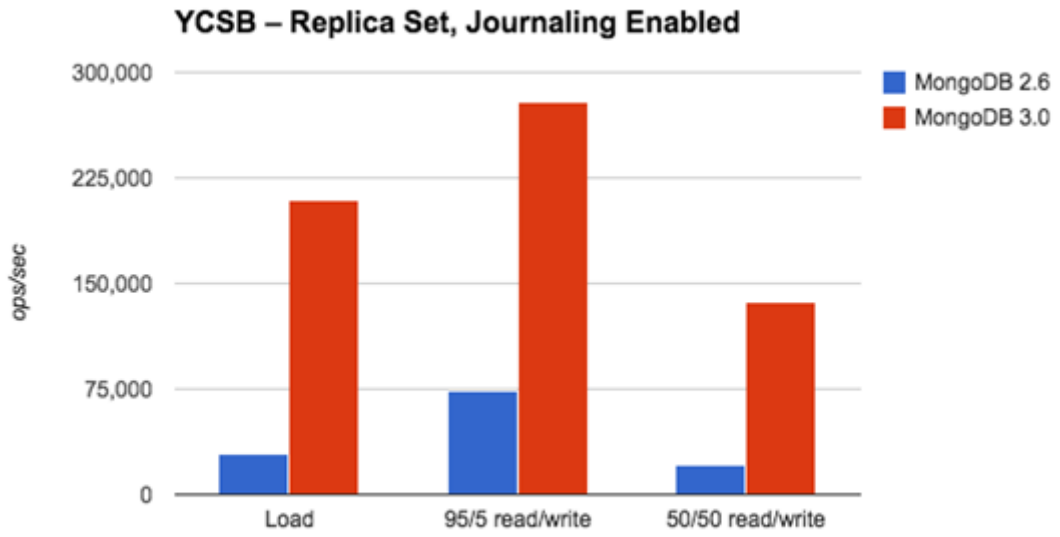
FROM sales	
GROUP BY ProductCategory, ProductSubCategory, ProductName	// Group By is handled by the emit(keys, values) line in the map() function above
	out : { inline : 1 } });

产品分类销售总额统计

4. Capped Collections比普通Collections的读写效率高

db.createCollection(“mycoll”, {capped:true, size:100000});
例：system.profile 是一个Capped Collection。
注意：
 固定大小；Capped Collections 必须事先创建，并设置大小。
 Capped Collections可以insert和update操作；不能delete操作。只能用drop（）方法删除整个Collection。
 默认基于 Insert 的次序排序的。如果查询时没有排序，则总是按照insert的顺序返回。
 FIFO。如果超过了Collection的限定大小，则用 FIFO **算法**，新记录将替代最先 insert的记录。

5. Mongodb 3.0.X版本性能较Mongodb 2.0.X有7-10倍提升，引入WiredTiger新引擎，同时支持MMAPv1内存映射引擎



注意：

 默认MMAPv1，切换至WiredTiger：mongod –dbpath /usr/local/mongodb/data –storageEngine wiredTiger

备注：若更换新引擎，则之前使用旧引擎建立的DB数据库无法使用。 建议先通过Mongodb的同步机制，将旧引擎建立的DB数据同步到从库， 且从库使用新引擎.

 选择 Windows 2008 R2 x64 或 Linux x64，Linux版本性能优于Windows，建议基于Linux系统进行架构选型

 根据RHEL版本号选择Mongodb相应Linux版本

 Mongodb Driver 与 Mongodb 版本一致

最后的建议

哪一种物理设计更适合Mongodb：范式化 & 反范式化 & 业务？

 范式化设计的思想是“完全分离”，存在关联查询，查询效率低，但写入、修改、删除性能更高

 反范式化设计的思想是“数据集中存储”，查询效率高，而Mongodb对查询机制支持较弱，看似成为一种互补

下面我们来看一个图书信息DB表设计案例：

示例1：范式化设计

```
{
    "_id" : ObjectId("5124b5d86041c7dca81917"),
    "title" : "MongoDB性能调优",
```

```
    "author" : [
            ObjectId("144b5d83041c7dca84416"),
            ObjectId("144b5d83041c7dca84418"),
            ObjectId("144b5d83041c7dca84420"),
    ]
}
```

分析：更新效率高，因为不需要关联表操作。比如更新作者年龄，只需要更新作者信息1张表就可以了。而查询效率低，因为需要关联表操作。比如查看某本图书的作者简介，需要先查图书信息表以获取作者ID，再根据ID，在作者信息表中查询作者简介信息。

示例2：反范式化设计

```
{
    "_id" : ObjectId("5124b5d86041c7dca81917"),
    "title" : "MongoDB性能调优",
    "author" : [
            {
                    "name" : "张三"
                    "age" : 40,
                    "nationality" : "china",
            },
            {
                    "name" : "李四"
                    "age" : 49,
                    "nationality" : "china",
            },
            {
                    "name" : "王五"
                    "age" : 59,
                    "nationality" : "china",
            },
    ]
}
```

分析：将作者简介信息嵌入到图书信息表中，这样查询效率高，不需要关联表操作。依然是更新作者年龄，此时更新效率就显得低，因为该作者出过多本书，需要修改多本图书信息记录中该作者的年龄。

示例3：不完全范式化设计

```
{
    "_id" : ObjectId("5124b5d86041c7dca81917"),
    "title" : "MongoDB性能调优",
    "author" : [
        {
            "_id" : ObjectId("144b5d83041c7dca84416"),
            "name" : "张三"
        },
        {
            "_id" : ObjectId("144b5d83041c7dca84418"),
            "name" : "李四"
        },
        {
            "_id" : ObjectId("144b5d83041c7dca84420"),
            "name" : "王五"
        },
    ]
}
```

分析：其实我们知道某本书的作者姓名是不会变化的，属于静态数据，又比如作者的年龄、收入、关注度等，均属于动态数据，所以结合业务特点，图书信息表肯定是查询频率高、修改频率低，故可以将一些作者的静态数据嵌入到图书信息表中，做一个折中处理，这样性能更优。

总结：Mongodb性能调优不是最终或最有效的手段，最高效的方法是做出好的物理设计。而什么样的物理设计适合Mongodb，最后还是由当前业务及业务未来发展趋势决定的。最后送给大家一句话“好的性能不是调出来的，更多是设计出来的”！