

Mongodb 针对复杂查询的特性

场景：计费话单查询

计费话单条件查询

```
2019 - 02 - 22T09 : 32 : 16.777 + 0800 I COMMAND[conn163]command gzbsms.ucCallRecord command : find {
  find : "ucCallRecord",
  filter : {
    $and : [{
      tenementId : {
        $in : ["880001"]}}, {calling : /^.*290727.*$/}, {called : /^.*1369.*$/}, {direction : "in"}, {trunkId : "SIPT_4_880001"},
        {wireNumber : /^.*0551.*$/}, {beginTime : {$gte : 1548000000000}}, {beginTime : {$lte : 1550764799000}}
    ]},
    sort : {
      beginTime : -1
    },
    limit : 15
  },
  planSummary : IXSCAN {
    direction : 1
  }
}
keysExamined : 3511273 docsExamined : 3511273 hasSortStage : 1 cursorExhausted : 1 numYields : 27508 nreturned : 0 reslen : 107 locks : {
  Global : {acquireCount : {r : 55018},acquireWaitCount : {r : 114},timeAcquiringMicros : {r : 136482}},
  Database : {acquireCount : {r : 27509}},
  Collection : {acquireCount : {r : 27509}}
}
protocol : op_query 20453ms
```

如上图所示:

- 查询条件: 企业ID: 880001, 主叫: 290727, 被叫: 1369, 入局, 中继: cop, 外线号: 0551, 时间范围: 2019-01-21 -- 2019-2-21
- 分页数量为 15条
- 扫描数据量为3511273 个文档, 3511273 个索引
- 查询到的数据为 0 条
- 耗时 20.45秒

查询慢的原因

- 查询条件多, 需要过滤的文档多, 导致扫描数据大
- 没有命中排序索引, 需要排序的数据量大
- 索引选举耗时的耗时

优化思路:

- 分表: 尽可能地使扫描数据记录减少
- 分词: 把模糊查询优化成, 分词匹配, 确保索引能够精确命中 (模糊查询会扫描所有的索引记录)
- 保证内存空间: 确保索引都在内存中扫描

针对 mongodb 的查询效率优化总结

- 扫描索引和文档越多, 耗时也越多
- 机器内存大小, 影响查询效率

优化建议如下：

查询数量优化

- 使用近似值（explain 中预测扫描行的数量）
- 简到优化（总数 - 已知小分数据的数据）
- 覆盖索引优化
- 汇总表/外部缓存数量（存储数量的数据）

对于复杂的模糊查询

- 索引优化策略

- [创建索引以支持查询](#)

当索引包含了查询的所有键时，索引可以支持该查询。创建可以支持查询的索引会带来极大的查询性能提升。

- [使用索引来排序查询结果](#)

为了支持高效的查询，当您需要指定被索引键的排列顺序和排序顺序时，请使用此处的策略。

- [确保索引与内存相适应](#)

当您的索引可以整个存储于内存时，系统可以避免从磁盘读取索引，这时您的处理过程会变得更快速。

- [创建能确保选择力的查询](#)

选择力是查询使用索引来缩窄结果集范围的能力。选择力使得MongoDB可以使用索引来完成匹配查询过程中的更多工作

- 排序优化

- 按照指定索引顺序进行排序

- 尽量做到读写分离

- 索引，集合加载到内存

- 预加载数据或者索引到内存: `db.runCommand({ touch: "collectionName", data: [true|false], index: [true|false] })`

- 查询方式优化

- 使得索引命中率提升
 - 缩小查询数据结果集的范围
 - 模糊查询尽量使用字段数据分词/左前缀索引匹配
 - 业务分离，能抽出来的业务，尽可能少去查询 数据量大的数据表

- 数据结构优化

- 如查询方式优化
 - 分表
 - 分库

计费话单优化方案

1 查询方式优化

查询条件优化（呼叫，被叫，外线号码 三选一查询）

原因：因为三个字段都是单独的索引，减少索引选举和文档比较的次数

2 索引调整

旧索引列表

```
{
  "_id_" : 242565120,
  "callRecord_tenementId_beginTime_endTime_idx" : 347521024,
  "callRecord_beginTime_endTime_tenementId_idx" : 536272896,
  "tenementId" : 82497536,
  "direction" : 83247104,
  "trunkId" : 82489344,
  "monthSetId" : 83300352,
  "callRecord_direction_trunkId_idx" : 85852160,
  "callRecord_beginTime_tenementId_idx" : 261197824
}
```

优化后的索引

```
{
  "_id_" : 437186560,
  "monthSetId_1" : 448692224,
  "beginTime_-1" : 526671872,
  "duration_1_beginTime_-1" : 244002816,
  "trunkId_1_beginTime_-1" : 148099072,
  "calling_arr_1_beginTime_-1" : 149823488, // 主叫分词之后的数组
  "called_arr_1_beginTime_-1" : 448135168, // 被叫分词之后的数组
  "wireNumber_arr_beginTime_-1" : 147931136, // 外线号码叫分词之后的数组
}
```

3 模糊查询字段进行分词

```
* **分词后的数据结构**
* **使用查询前缀查询**（默认索引前缀匹配）
```

```
{
  "_id" : ObjectId("5b23db5faeee1022a08b8e8a"),
  "_class" : "com.ejiahe.bms.bean.mongobean.CallRecord",
  "tenementId" : "880001",
  "callingNumID" : "249999",
  "calledNumID" : "4444",
  "callingUserId" : "u110002",
  "calledUserId" : "",
}
```

```

    "direction" : "inner",
    "beginTime" : NumberLong("1529076539000"),
    "endTime" : NumberLong("1529076551000"),
    "duration" : 12,
    "billUserId" : "u110002",
    "calledRes" : "OK",
    "calling" : "249999(249999)",
    "called" : "4444",
    "wireNumber": "12345",
    "calling_arr": ['24', '99', '2499', '24999', '249999'],
    "called_arr": ['44', '444', '4444'],
    "wireNumber_arr": ['12', '23', '34', '45', '123', '234', '345', '1234', '2345', '12345'],
    "durationTime" : "00:00:12",
    "createTime" : NumberLong("1529076575673")
}

```

4 按月分表

- 按月分表查询(分页尽量避免跨月操作)
- 根据时间进行分页查询，减少扫描数据量

5 保证足够内存

- 当前德邦 `mongodb` 内存中的数据状况

```

{
    "bits" : 64,
    "resident" : 17660,           // 17G 物理内存 单位 M
    "virtual" : 22261,           // 20G 虚拟内存 单位 M
    "supported" : true,
    "mapped" : 0,
    "mappedWithJournal" : 0
}

```

- 当前德邦 `mongodb` 实际存储数据状况

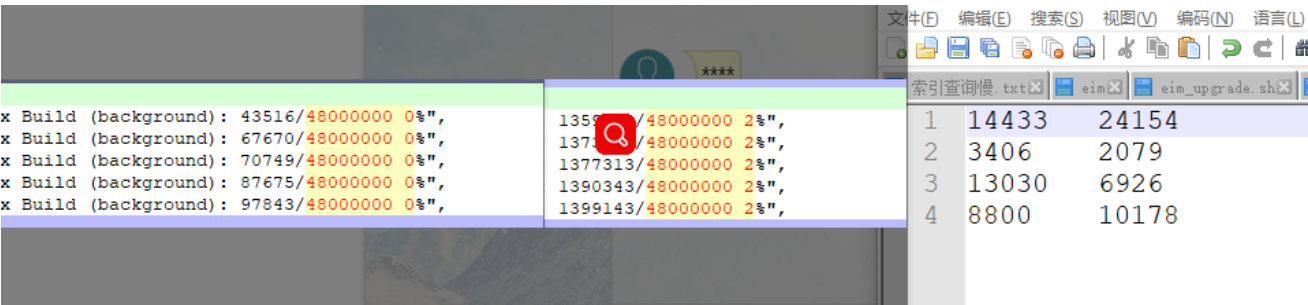
```

{
    "db" : "gzbsms",
    "collections" : 20,
    "views" : 0,
    "objects" : 26362385,
    "avgObjSize" : 845.233533119253,
    "dataSize" : 22282371815,     // 20G 数据的数据大小 单位 b
    "storageSize" : 6578077696,   // 6.5G 磁盘中文件大小 单位 b
    "numExtents" : 0,
    "indexes" : 59,
    "indexSize" : 1806274560,     // 1.6G 索引的大小 单位 b
    "ok" : 1
}

```

左缀索引和分词比较（基于本地数据库进行比较）

建立索引速度



左边为 `callingarr_endTime` , 数组 + 结束时间

右边 `calling_beginTime`

结果显示：索引创建结果相差不大

数据准备

- 左前缀：无需修改数据，只是添加索引即可
- 分词：需要额外，对模糊查询的字段进行分词，然后当作做一个数组的查询字段（当前测试版本是使用js脚本进行插入的，一百万一次会比较慢）；当前本地库已经插入240万条带有calling 分词的数据

JS执行代码

```
// 获取指定长度的数字随机数
function RndNum(n){
    var rnd="";
    for(var i=0;i<n;i++){
        rnd+=Math.floor(Math.random()*10);
    }
    return rnd;
}

// 数组元素去重
Array.prototype.push2 =function(args){

    if(this.indexOf(args) == -1){
        this.push(args);
    }
};

// 数组元素进行分词
Array.prototype.splitword=function(str,len){
```

```

    if(len <= 0) len = 0;
    var size = str.length;
    if(size == 0) return;

    if(len >= size + 1) return;

    var temLen = 0;

    for(;temLen < size; temLen++){
        if(temLen + len > size){
            continue;
        }

        var ele = str.substring(temLen,temLen + len);
        this.push2(ele);
    }

    this.splitword(str,len + 1);
};

// 执行分词插入
db.ucallRecord.find().forEach(function(item){
    var arr1 = new Array();
    arr1.splitWord(RndNum(11),1); // 11 位字符进行分词插入
    db.ucallRecord.update({_id:item._id},{ $set:
    {calling:arr1,called:arr1,wireNumber:arr1}}, {upsert:true});
});

```

分词java 代码

```

public static Set<String> splitToSet(String str,int len){

    if(len <= 0) len = 0;

    Set<String> set = new LinkedHashSet<>();

    int size = str.length();
    if(size == 0) return set;

    if(len == size + 1) return set;

    int temLen = 0;
    for(;temLen < size; temLen++){

        if(temLen + len > size){
            continue;
        }

        set.add(str.substring(temLen,temLen + len));
    }
}

```

```
set.addAll(splitToSet(str,len + 1));  
return set;  
}
```

文档大小限制

- 集合的最大所有个数：64
- 索引名称长度：包括数据库于集合名称总共不超过125字符。
- 联合索引最大字段个数：31
- 单个文档大小不能超过16M

查询速度比较

记录表信息：

Key	Value
ns	gzbsms.ucCallRecord
size	27,471,725,878 (25.6 GiB)
count	48,000,000 (48.0 M)
avgObjSize	572
storageSize	5,149,343,744 (4.8 GiB)
capped	false
wiredTiger	{ 14 fields }
nindexes	12
totalIndexSize	9,981,358,080 (9.3 GiB)
indexSizes	{ 12 fields }
id	437,186,560 (437.2 M)
monthSetId_1	448,692,224 (448.7 M)
calling_1	1,355,358,208 (1.4 G)
called_1	1,634,295,808 (1.6 G)
beginTime_-1	526,671,872 (526.7 M)
duration_1	244,002,816 (244.0 M)
trunkId_1	148,099,072 (148.1 M)
wireNumber_1	1,327,144,960 (1.3 G)
callingarr_1	950,657,024 (950.7 M)
callRecord_beginTime_tenementId_id	909,975,552 (910.0 M)
callRecord_direction_trunkId_idx	195,133,440 (195.1 M)
callingarr_1_beginTime_1	1,804,140,544 (1.8 G)
ok	1

查询条件

左前缀查询

```
db.ucCallRecord.find({calling:
{$regex:/^12345.*/}}).sort({"beginTime":-1}).limit(100).explain('executionStats');
```

分词查询

```
db.ucCallRecord.find({callingarr:'12345'}).sort({"beginTime":-1}).limit(100).explain('executionStats');
```


查询条件	左前缀	分词
12345	索引扫描: 353, 文档扫描: 94, 耗时: 3 ms, 返回: 94	索引扫描: 77, 文档扫描: 77, 耗时: 1 ms, 返回: 77
123	索引扫描: 64900, 文档扫描: 13738, 耗时: 974 ms, 返回: 100	索引扫描: 100, 文档扫描: 100, 耗时: 23 ms, 返回: 100
12	索引扫描: 789842, 文档扫描: 147286, 耗时: 6521 ms, 返回: 100	索引扫描: 100, 文档扫描: 100, 耗时: 33 ms, 返回: 100
1	索引扫描: 8814688, 文档扫描: 1073952, 耗时: 77368 ms, 返回: 100	索引扫描: 100, 文档扫描: 100, 耗时: 1 ms, 返回: 100

测试小结:

左前缀匹配索引方式进行模糊查询

优点

- 无需做业务上的修改, 后期开发灵活, 只需要查询时使用正则表达式即可
- 在查询越精确的数据速度越快, 若没有查询到匹配的记录会返回很快, 基本上都是毫秒级别
- 内存, 存储压力较少

缺点

- 当匹配到的记录越多, 返回数据越慢 (因为需要对查询到的记录进行排序)

分词方式进行模糊查询

优点

- 数据按照索引顺序, 进行精确匹配, 查询速度很快

缺点

- 空间换时间: 数据大小增加
- 逻辑复杂: 需要对模糊匹配字段进行分词, 同时需要添加数组索引, 在逻辑复杂度增加
- 操作不灵活: 当需求变更是, 若添加/删除模糊字段, 需要重新定义数据结构
- 若是多个模糊查询需要命中到索引, 则会扫描匹配分词数组, 测试若分词数量大时, 查询会相对的慢很多
 - 如图:

Key	Value	Type
➤ (C) (1)	{ 11 fields }	Object
ns	gzbmsms orders1	String
size	6,905,439,476 (6.4 GiB)	Int32
count	7,883 (7.9 K)	Int32
avgObjSize	875,991 (855.5 KiB)	Int32
storageSize	1,334,431,744 (1.2 GiB)	Int32
capped	false	Bool
▷ (C) wiredTiger	{ 14 fields }	Object
nindexes	4	Int32
totalIndexSize	3,336,814,592 (3.1 GiB)	Int32
➤ (C) indexSizes	{ 4 fields }	Object
id	126,976 (127.0 K)	Int32
c_1_a_-1	1,111,814,144 (1.1 G)	Int32
b_1_a_-1	1,112,645,632 (1.1 G)	Int32
d_1_a_-1	1,112,227,840 (1.1 G)	Int32
ok	1	Int32

	_id	a	b	c	d
1	ObjectId("5c7cc644ea4b6983b46801ba")	1.0	Array[3]	Array[3]	Array[3]
2	ObjectId("5c7cc645ea4b6983b46801bb")	2.0	Array[3]	Array[3]	Array[3]
3	ObjectId("5c7cc645ea4b6983b46801bc")	3.0	Array[3]	Array[3]	Array[3]
4	ObjectId("5c7cc9e6ea4b6983b46801be")	0.0	Array[5951]	Array[5951]	Array[5951]
5	ObjectId("5c7cc9e6ea4b6983b46801bf")	1.0	Array[5964]	Array[5964]	Array[5964]
6	ObjectId("5c7cc9e6ea4b6983b46801c0")	2.0	Array[5956]	Array[5956]	Array[5956]
7	ObjectId("5c7cc9e6ea4b6983b46801c1")	3.0	Array[5953]	Array[5953]	Array[5953]
8	ObjectId("5c7cc9e6ea4b6983b46801c2")	4.0	Array[5960]	Array[5960]	Array[5960]
9	ObjectId("5c7cc9e7ea4b6983b46801c3")	5.0	Array[5961]	Array[5961]	Array[5961]
10	ObjectId("5c7cc9e7ea4b6983b46801c4")	6.0	Array[5961]	Array[5961]	Array[5961]

2	db.orders1.find({b:'2',c:'994',d:'511'}).sort({a:1}).limit(10).explain('executionStats');
3	
4	
0.103 s	
Key	Value
"0"	[MinKey, MaxKey]
executionStats	{ 6 fields }
executionSuccess	true
nReturned	10
executionTimeMillis	91
totalKeysExamined	65
totalDocsExamined	65
executionStages	{ 4 fields }

分词数据兼容问题

- 在系统升级之前，使用程序对文档进行分词补充，可以写要专门的程序用来做分词更新的库
- 查询兼容：
 - 记录在分词插入数据前的最后一条记录的创建时间（或者一个升序的唯一标识）
 - 标识此创建时间之前的使用 不适用分词查询， 创建时间之后的使用分词查询

针对查询方式优化建议：

- 查询条件尽可能简单（不建议条件组合太复杂）
- 模糊查询并且数据量（1000w以上）时建议： 查询字段的值进行分词
- 查询数量：大数据查询数量是相对慢的，见： [查询数量优化](#)
- 排序字段若是查询条件时建议选为必填字段
- 尽可能减少两个以上的模糊查询字段同时查询，
- 模糊查询字段长度需要控制，具体长度需和研发讨论

附录

[mongodb 官方文档](#)