话单查询慢优化

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 (**模糊查询导致**)
- 条件索引多，选举耗时的耗长
- 内存不够

优化思路:

- 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
- 分表/分库： 尽可能地使扫描数据记录减少

模糊优化方案

- 最左前缀匹配
- 模糊查询字段分词匹配

环境:

数据： 2700W条

分词查询环境: 192.168.76.82

模糊查询环境: 192.168.76.84

左前缀查询环境: 192.168.76.84

192.168.76.84：数据库占用内存: 8.1G, 虚拟内存: 8.4G ucCallRecord: 数据大小 6.4G 压缩文件大小：1.9G 索引大小：13.4G

192.168.78.30：数据库占用内存: 7.1G, 虚拟内存: 7.4G ucCallRecord: 数据大小 21.5G 压缩文件大小：6.5G 索引大小：23G,

查询条件

模糊查询

```
db.ucCallRecord.find({
    tenementId : {
        $in : ["880001"]
    },
    calling : {
        $regex : /^.*290727.*$/
    },
    direction : "in",
    trunkId : "SIPT_4_880001",
    beginTime : {
        $gte : 1548000000000,
        $lte : 1550764799000
    }
}).sort({
    beginTime : -1
}).limit(15).explain('executionStats');
```

左前缀查询

```
db.ucCallRecord.find({
    tenementId : {
        $in : ["880001"]
    },
    calling : {
        $regex : /^290727.*$/
    },
    direction : "in",
    trunkId : "SIPT_4_880001",
    beginTime : {
        $gte : 1548000000000,
        $lte : 1550764799000
    }
}).sort({
    beginTime : -1
```

```
}).limit(15).explain('executionStats');
```

分词查询

```
db.ucCallRecord.find({
    tenementId : {
        $in : ["880001"]
    },
    callingarr : '290727',
    direction : "in",
    trunkId : "SIPT_4_880001",
    beginTime : {
        $gte : 1548000000000,
        $lte : 1550764799000
    }
}).sort({
    beginTime : -1
}).limit(15).explain('executionStats');
```

查询条件	模糊查询	左前缀	分词
主呼叫：290727 企业：880001 方向：入局 中继：SIPT_4_880001	索引扫描：653772， 文档扫描：2， 耗时：1325 ms 返回：2	索引扫描：439078， 文档扫描：0， 耗时：1203 ms 返回：0	索引扫描：2034， 文档扫描：2034， 耗时：12 ms 返回：15
主呼叫：2907 企业：880001 中继：SIPT_4_880001	索引扫描：8539， 文档扫描：15， 耗时：14 ms 返回：15	索引扫描：6565， 文档扫描：15， 耗时：17 ms 返回：15	索引扫描：16， 文档扫描：16， 耗时：1ms 返回：15
主呼叫：290 中继：SIPT_4_880001	索引扫描：8539， 文档扫描：15， 耗时：22ms 返回：15	索引扫描：6565， 文档扫描：15， 耗时：22 ms 返回：15	索引扫描：16， 文档扫描：15， 耗时：0ms 返回：15
主呼叫：29	索引扫描：12670， 文档扫描：15， 耗时：116ms 返回：15	索引扫描：8344， 文档扫描：15， 耗时：17ms 返回：15	索引扫描：15， 文档扫描：15， 耗时：0 ms 返回：15

查询条件	模糊查询	左前缀	分词
主呼叫：290727 企业：880001 方向：入局 中继： SIPT_4_880001	索引扫描： 653772， 文档扫描：2， 耗时：1325 ms 返回：2	索引扫描： 439078， 文档扫描：0， 耗时：1203 ms 返回：0	索引扫描：2034， 文档扫描：2034， 耗时：17 ms 返回：15
主呼叫：2907 企业：880001 方向：入局 中继： SIPT_4_880001	索引扫描：28159， 文档扫描：15， 耗时：56 ms 返回：15	索引扫描： 439078， 文档扫描：0， 耗时：1084 ms 返回：0	索引扫描：2377， 文档扫描：2377， 耗时：20 ms 返回：15
主呼叫：290 企业：880001 方向：入局 中继： SIPT_4_880001	索引扫描：3707， 文档扫描：3707， 耗时：8ms 返回：15	索引扫描： 439080， 文档扫描： 439080， 耗时：1060 ms 返回：2	索引扫描：2570， 文档扫描：2570， 耗时：12 ms 返回：15
主呼叫：29 企业：880001 方向：入局 中继： SIPT_4_880001	索引扫描：485， 文档扫描：15， 耗时：1ms 返回：15	索引扫描： 372989， 文档扫描：15， 耗时：948 ms 返回：15	索引扫描：338， 文档扫描：338， 耗时：7 ms 返回：15

测试小结：

模糊查询

优点

- 空间占用小
- 查询数据准确，全部走索引
- 查询速度 指标: 扫描： 3801900条索引 需要 7614ms即可查寻出来， 当前机器内存为： 8G,

缺点

- 随着数据量增大，查询会慢
- 单线程插入速度： 插入10w 条数据需要 884s， 换算下来： 113条/s
- 因为全部都是覆盖索引的原因，导致每条查询语句都需要指定索引查询，开发成本高，程序业务应变能力差

左前缀匹配索引方式进行模糊查询

对于分表使用覆盖索引之后，已经没什么优势可言， 加以选择模糊查询的方式进行查询

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,所需要内存增大（若内存不足可能会很大程度上影响查询速度）
- 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
- 单条数据体积较大，若是数据不再内存，且需要比较的数据量大时，查询会慢
- 若是有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（**建议不要同时使用两个以上的模糊查询**）

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）对于大数据查询，条件越少越易于查询**
- **分词使用场景: 分词字段尽量小**
  - mongodb 一行数据预分配空间： 21b, 列预分配空间： 11b , 中文：3b , 字母/数字：1b , 占位符: 1b
  - mongodb 数组据预分配空间： 21b, 列预分配空间： 12b , 中文：3b , 字母/数字：1b , 占位符: 1b，一个元素占用空间 7b
  - ```
    db.test1.insert({test:"唐诗雅"});
    42 = 21 + 11 + 3*3 + 1

    db.test2.insert({test:["唐","诗","雅","唐诗","诗雅","唐诗雅"]});
    111 = 21 + 12 + (7)* 6 + (3 + 1)*3 + 2 *( 3*2 + 1) + 3*3+1

    db.test1.insert({test:"唐诗雅",test1:["唐","诗","雅","唐诗","诗雅","唐诗雅"]});
    132 = 21 + (11 + 3*3 + 1) + (12 + (7)* 6 + (3 + 1)*3 + 2 *( 3*2 + 1) + 3*3+1)
    ```
  -
- **当数据量大时， 在设计时能否针对数据查询进行分类，减少全表的模糊查询如: 外线号码，主叫号码，入叫号码为等值查询， 名字为模糊查询**
- **复杂查询数量尽可能地减少搜索数据全量搜索， 区分热点数据 和全量数据查询**
    ```