# adhoc项目这一年走过的路,以及我们的进步

本文还没有最终整理完成，错别字很多，有不妥之处敬请见谅。

|  |  |
| --- | --- |
| 指标 | 描述 |
| 总数据量 | 原先：2亿+ 现在400亿+  400亿是目前的真实值，实际上现在的10台机器，内存和磁盘都非常富裕，预计千亿内没问题。 |
| 单次查询扫描的数据量 | 原先：2亿+ 现在 小于30亿  实际现在mdrill是没限制，但是扫描的速度跟数据量成正比，30亿大约需要30~120的返回时间，超过这个数值对于adhoc项目来说是无意义的故一直没有放开。 |
| 对内存的消耗 | 降低到原来的1/8,设置1/16,1/32或更多   1. 将lucene默认的将整个列数据全部都load到内存中的方式修改为load 每个列的值的编码代号，操作的时候也仅仅操作这些代号，真正展现的时候再将这些编号转换成真实的值，编号的数据类型根据某个列的值的重复程度可以为byte,short,int 2. 将数据进行分区(默认按照时间)，用到的数据会加载到分区中，不用的分区会从内存中踢出，采用LRU的方式管理，如果同时需要检索大量的分区，则进行排队处理，一个分区一个分区的处理。 3. 多个表之间也合并，共享内存，用到的表才会加载到内存中，没用到的则在硬盘中存储。 4. 原先merger server与shard是在同一个进程中的，每次查询的时候随机使用其中一个shard作为merger server,如果每次查询merger server使用1G的内存，但shard的数量非常多，merger server每次只用一个，但是为每个shard都额外分配1G内存就是浪费，新版mdrill将这两者分开，避免浪费。 5. 按照内存大小进行LRU，而不是按照field的个数，不同列因为重复读不同对内存的消耗也不一样，按照个数lru不合理，按照总内存使用LRU 6. 由于每次逆旋都需要消耗时间，当lur被淘汰的时候，将逆旋的结果保留到硬盘中，以备下次使用。 |
| 查询接口 | 原先：Higo-common 现在：标准SQL  Higo-common的使用复杂性就不多说了  Sql的使用文档  https://github.com/muyannian/higo/blob/master/doc/hsql.docx?raw=true |
| 创建表 | 原先：  1.编辑schema.xml  2.配置solr目录，并上传到server  3.登陆后台配置表，机器名，如何分组，分发  4.点击部署  现在：  通过create table 语句创建表  对于我们的应用场景，建表语句并不需要手工编写，adhoc项目后台都会对应一个hive表，通过hive的表明是可以直接得到create table的语句的。 |
| 集群可扩展性 | 原先：10~30台 现在：10台~百台   1. 旧版的mdrill使用单层次的merger server,来合并每个shard的结果，由于每个shard都会1W个结果，60个shard就是60W，如果600个shard内，单个merger server意味着在机器的扩展上面是一个瓶颈。通常根据cpu和内存情况一台机器会使用6~10个shard,30台机器以为着是180~300个shard这对旧版来说已经不可接受了。 2. 新的mdrill使用多层次的merger server,且优先合并同一台机器的shard,这样随着机器的扩容，这些结果合并的压力会分散到多个不同的merger server中去。 3. 旧版的mdrill每次部署索引都从某一台固定的机器上通过scp的方式将程序copy下来，由于程序一般都比较大，scp将来也会是瓶颈。 |
| 简化部署，统一配置 | 原先   1. schema.xml要配置 2. solr的一些配置 3. 要登陆后台分配存储目录，端口，分发 4. higo.properities要配置好多参数 5. ssh无密码登陆要配置 6. higo-config.sh要配置 7. higo-core.sh要配置 8. 要编译程序后，将生成的dist放置到不同的目录（每个表一个目录，每次更新都要全部更新） 9. tomcat下也要配置jar和配置config，分发的目录要一定要注意别copy错了 10. 启动的时候要注意启动的路径，不然无法读取配置文件 11. 复杂的启动参数 12. 如果有程序改动或配置文件变更，又要重头来一次   现在  唯一一个配置文件storm.yaml  配置的参数个数也减少了很多也统一了。   1. 新创建一个表像我这样一个熟手，不折腾个2~3天根本就搞不定，每次改配置，那更悲剧，好多目录啊好多配置文件啊，同一个配置文件要分发到好多目录（还好当时就2台机器^\_^）   新版的部署  <https://github.com/alibaba/mdrill/blob/master/doc/INSTALL.docx?raw=true>  新版的重启，更简单  更改配置后 stop 然后在启动即可 |
| 监控与容错 | 原先   1. 简单的任务启动页面，之后任务挂了就挂了，也没有任何的过载保护措施 2. 任务启动成功失败与否没有判断 3. 在部署索引的时候，由于索引下载时间较长，需要停止5~20分钟的服务，重启solr   现在   1. 通过心跳机制监控任务的健康状态，如果任务异常或硬盘故障会重启和更换硬盘，如果机器宕机，任务会迁移到其他机器 2. 任务的内存，端口，进程ID，心跳时间，表的数据量，起始时间，总记录数，对应的分区都记录 3. 添加过载保护，如果并发特别大，则会排队，限制并发。 4. 部署索引的时候并不影响服务,不重启solr   其他   1. 部署索引先下载到本地,然后通过rename的方式切换索引，改造solr   ,切换索引的时候不需要重启solr   1. 任务的调度与分配交由蓝鲸系统管理，之前的higo调度(higo-manager)：采用shell通过ssh无密码登陆的方式部署与管理 |
| 创建索引优化 | 原先   1. 每天创建索引的时间超过10个小时 2. 每天全部的数据都参与build,每次部署都要分发全量的索引,IO比较大 3. 索引的大小为原始数据的3倍 4. 仅支持文本文件创建索引 5. 创建索引不能使用大的hadoop集群，必须使用高配置的有大硬盘的小集群   现在   1. 每天创建索引的实际为30~60分钟 2. 采用增量索引，每天只处理增量的数据 3. 索引的大小小于原始数据的大小 4. 创建索引支持多种文件格式与分隔符   如何做的   1. 不分词的field不存储，所有的原始数据都通过压缩比较高的tis与frq文件中获取 2. lucene修改:可以直接在hdfs中创建 3. lucene修改:addIndexesNoOptimize优化，提升并发 4. 通过addIndexesNoOptimize优化与数据的分区特性实现增量索引   详细的作用和解释请参考  <https://github.com/muyannian/higo/wiki/Lucene> |
| 功能提升 | 原先   1. 只支持少量组数的分组统计汇总（如果组数超过50，那么将会有性能瓶颈） 2. 分类汇总后无法进行排序 3. 不支持count(distinct)查询 4. 不支持null列的统计 5. 含有特殊字符的组会存在异常报错   现在   1. 支持任意维度的分组统计（对于组数没限制，1w组内为准确排序，超过1W为近似排序） 2. 分类汇总后可以按照某个统计值，或列进行排序 3. 支持近似count(distinct)查询 4. 支持任意列小表join 5. 支持null稀疏列的统计与汇总 6. map端任意列join(sql中未体现) 7. 特殊字符不在受影响   如何做的   1. 旧版的mdrill的查询是使用solr的facet，每次只能获取单个列的group count,如果要进行多个列的group by,则需要分别查询每个列的facet,,   然后将这些facet的只进行笛卡尔乘积，组合成每总可能存在的组合，  然后将这些组合并发的化作很多小的子查询，计算没一个组的统计值。  如果要同时进行三个维度的group by，对另外3个维度进行统计汇总，如果每个维度有50个不同的值那么需要进行50\*50\*50\*3个统计维度=  =125000次查询才能得到每个列的统计值，真实情况是不可能实现的，意味着也就不能进行排序  如果某个field的值含有特殊字符，由于需要转移，在进行子查询的时候经常出现报错的情形  新版mdrill的实现请参考如下这些链接  <https://github.com/muyannian/higo/wiki/distinct>  <https://github.com/muyannian/higo/issues/117>  <https://github.com/muyannian/higo/issues/117>  <https://github.com/muyannian/higo/issues/118>  <https://github.com/muyannian/higo/issues/119>  <https://github.com/muyannian/higo/issues/95> |
| Java Gc优化 | mdrill2，为减少gc次数，做了需要改进其中有  1,许多地方迭代使用的临时对象不需要反复声明，共用即可  2.各种block cache的实现 |