ClickHouse使用技巧和注意点总结

1. 选择合适的分区规则

对于MergeTree，在CK的存储是根据根据分区存储的，也就是“分区/列”，选择合适的分区可以在查询的时候忽略掉一些文件夹。

2. 写入之前事先进行排序，这样可以降低数据导入后 ClickHouse 后台异步 Merge 的时候涉及到的分区数

MergeTree在创建的时候必须要要求order by，在插入数据的时候不要求顺序，CK会在插入之后后台异步的去合并，让最终的数据在每个分区内是有序的，如果实现排序了，那么后台合并树的时候就会减少压力。

3. 建议每次写入不少于1000行的批量写入，可以使用多个INSERT进行并行写入，这将带来线性的性能提升

MergeTree在写入数据的时候，会新建一个分区，插入1行是一个分区，插入1000行也是一个分区。

4. JOIN时大表在左边

这是因为CK会把右边的表全部读到内存中去。

5. 学会使用explain查看执行计划

> 这里参考https://cloud.tencent.com/developer/article/1662230的例子

在CK20.6之后，就支持用explain查看执行计划了。

```
EXPLAIN [ PLAN | AST | SYNTAX | PIPELINE ] [setting = value, ...] SELECT ...
```

```
EXPLAIN PLAN SELECT 1:
Union
  Expression (Projection)
    Expression (Before ORDER BY and SELECT)
      ReadFromStorage (Read from SystemOne)

4 rows in set. Elapsed: 0.001 sec.
```

PLAN 是 EXPLAIN 的默认值，所以 PLAN 修饰词可以省略。

EXPLAIN PLAN 目前还可以设置 3 个参数，它们分别是：

header，打印计划中各个步骤的 head 说明，默认关闭;

description，打印计划中各个步骤的描述，默认开启；

actions，打印计划中各个步骤的详细信息，默认关闭。

> 如果遇到的是老版本的CK，可以参考这个博客https://cloud.tencent.com/developer/article/1604964?from=article.detail.1662230

6. 避免使用select *

其实不止ClickHouse应该避免select *，大部分数据都不推荐使用select *

首先，毫无疑问的，不需要的列会增加数据传输时间和网络开销，尤其是一些varchar这样的大字段，会影响IO操作。

其次，select *对于mysql而言，会使覆盖索引策略失效，也就是本身直接从索引能拿到数据的，就不需要再读内存了。对于CK而言，则更为直接，本身列式存储，要哪一列就去查哪列，查询不想干的列完全就是多余的开销。

7. 如果程序需要使用大量JOIN，可以借助上层应用侧的缓存服务，或使用JOIN表

JOIN操作结果不会缓存，所以每次JOIN操作都会生成一个全新的执行计划。如果应用程序会大量使用JOIN，则需进一步考虑借助上层应用侧的缓存服务或使用JOIN表引擎来改善性能（JOIN表引擎不支持ASOF精度）。JOIN表引擎会在内存中保存JOIN结果。

在某些情况下，IN的效率比JOIN要高。

在使用JOIN连接维度表时，JOIN操作可能并不会特别高效，因为右则表对每个query来说，都需要加载一次。在这种情况下，外部字典（external dictionaries）的功能会比JOIN性能更好。

8. 尽量不要delete和update，update/delete 的使用场景是一次更新大量数据，修改和删除很重，因为会复制一个分区，下次合并再删除。频繁单条更新可以使用ReplacingMergeTree

像携程就是使用的增量更新。

9. 维度表数据可以使用字数据字典，因为数据字典常驻内存，这样可以避免不必要的join

对于一个类似字典的表，比如id=1表示部门A，id=2表示部门B

10. wal机制

对于Mysql而言，wal机制可以使插入和更新操作先在内存执行，此时内存的数据被称为脏页，然后先写到redolog里，等后面有空闲了，再将内存中的数据刷回磁盘。

某天看到虎哥的博客里面说CK现在支持WAL，只需要设置下面这个参数：

```
SETTINGS: min_rows_for_compact_part=2
```

> 不过在CK的源代码MergeTree.h中看到这个设置暂时还是Experimental，怪不得我在官网的MergeTree下面没有看到。

执行 2 条写 SQL，数据会先写到 wal.bin 文件，当满足 `min_rows_for_compact_part=2` 后，merger 线程触发合并操作，再生成新的分区。

11. 合理配置配置config.xml和user.xml

比如上面那个setting，就可以直接写到配置文件里，作为默认的设置，省事儿！

12. CPU一般在50%左右会出现查询波动，达到70%会出现大范围的查询超时，CPU是最关键的指标，要非常关注
13. 可以使用prewhere

prewhere首先会读取指定的列数据，来判断数据过滤，等待数据过滤之后再读取select 声明的列字段来补全其余属性。

在某些场合下，prewhere语句比where语句处理的数据量更少性能更高。

不过这个CK会自己优化，所以其实不用特意去写prewhere。

14. optimize语句可以主动进行合并，但是不能频繁执行
15. 对于group by内存不够，可以启动max_bytes_before_external_group_by

设置确定倾销的阈值RAM消耗 `GROUP BY` 临时数据到磁盘，虽然这条语句估计还是执行得很慢，但是可以跑了。

16. 使用LowCardinality代替String

CK存储是用LZ4压缩的，不同的类型压缩的字节也不一样，LowCardinality比String压缩得更好，在数据分布千万的级别下少占用百分之30的空间，这可以加快读取数据的速度

17. 可以为数据配置冷热存储策略

ClickHouse默认策略将新写入数据存储为热数据，提供高效查询。当热数据存储量达到业务使用阈值时，自动将最早写入部分数据转为冷数据存储，将这部分热数据转化为冷数据，从而释放热数据存储空间。

除了磁盘阈值策略，还可以通过TTL策略，将高频查询数据放到数据存储中。

18. 选择合适的数据压缩方式

CK支持多种压缩算法，比如LZ4，LZ4HC，ZSTD，关于压缩算法的测试，可参考这篇文章

冷数据查询情况下（无OS缓存），LZ4和ZSTD区别不大，因为消耗在IO方面的时间，远大于消耗在数据解压缩上面的时间。

热数据情况下（有OS缓存），LZ4会更快，此时IO代价小，数据解压缩成为性能瓶颈。

至于LZ4HC，压缩率比不过ZSTD，压缩时间还不短。

19. 使用数据TTL

CK可以通过TTL管理数据声明周期，支持几种不同粒度的TTL：

1） 列级别TTL：当一列中的部分数据过期后，会被替换成默认值；当全列数据都过期后，会删除该列。

2）行级别TTL：当某一行过期后，会直接删除该行。

3）分区级别TTL：当分区过期后，会直接删除该分区。

注意点

1. 查询特定某一行不是CK的强项

前面已经提到了，CK的存储是按列存储的，查一行是CK最没有性价比的操作。

2. 不支持高并发，官方建议qps为100

我个人的理解是，CK在查询的时候本身就会极尽性能，采用对称多处理，默认使用一般的CPU，所以它不希望你再高并发的查询。

3. ClickHouse特别适合数据量大，查询次数可控的场景
4. ClickHouse的分布式表性能性价比不如物理表高，建表分区字段值不宜过多，防止数据导入过程磁盘可能会被打满
5. 不支持事务，不支持真正的删除/更新
6. CK的二级索引和普遍认为的不太一样

CK的二级索引是moloap的预聚合，而不是在普通字段上建立索引。

7. 更新和插入操作没有原子性

其他

> ClickHouse常见问题排查
>
> CK负载均衡chproxy