Raft log持久化机制和使用log进行Catch up

- 背景
- 目的
- 设计
    - raft log持久化机制
    - 数据写入
    - 数据读取
    - 重启恢复
    - 删除策略

背景

leader内存中保存了一定条数的log日志（默认是1000条），用于当follower落后leader很多的时候，可以使用日志进行追赶。当follower落后leader超过此阈值时候，只能使用snapshot进行追赶。但是由于snapshot有如下几个问题。使得发送snapshot的代价较高。

1. snapshot会在同一个raft组内发送flushPlan的日志，协调同一个raft组的其他节点同时刷盘，目的也是让同一个raft组的其他节点保持tsFile的version一致，用于判断是否是同一个文件。
2. 在执行flushPlan的时候，为了确保这个FlushPlan之后的日志不被应用(raft commit log index和term不被更新)，所以需要block住正常的请求。
3. follower会从leader拉取一些tsfile，这样花费时间也较大。

综上，snapshot是一个代价较高的操作。

考虑如下情况导致的follower落后leader的日志超过了leader内存中保留的日志条数阈值，都会导致leader发送snapshot，这样就会带来代价较高的操作。

1. follower由于gc(长达几秒钟)。
2. 网络暂时拥堵。

所以拟采用持久化log用于catchup。

目的

重新设计log持久化日志，当follower日志落后leader超过leader内存中保留的阈值的时候，采用从持久化raft log日志中进行读取，用raft log进行追赶。当然不可能什么情况下都用log进行追赶(比如新加入节点，或是follower落后leader很多了，用log进行追赶的代价可能就远远大于用snapshot进行追赶的代价了)，这就带来一个问题：当follower落后leader多少条日志的时候，可以使用log进行追赶？超过此阈值，就使用snapshot进行追赶。

设计

raft log持久化机制

备注：内存中log删除机制不变：会有一个最小值和最大值，以及一个定时查看机制。每次数据写入都会判断内存中log是否到达最大值，如果达到则就会进行删除，保留最小值数量的log。同时定时机制也会定时去看内存中log是否超过最小值，如果是则就进行删除。目前一个issue考虑log内存占用(https://issues.apache.org/jira/browse/IOTDB-854)，防止内存log占用超过一定内存导致OOM。

当新的raft log到来，除了保存到内存中之外，也会保存到磁盘上，log在磁盘上的格式如下所示：

一个raft log index文件对应一个raft log data文件。

raft log index 内存中保存形式是如下：

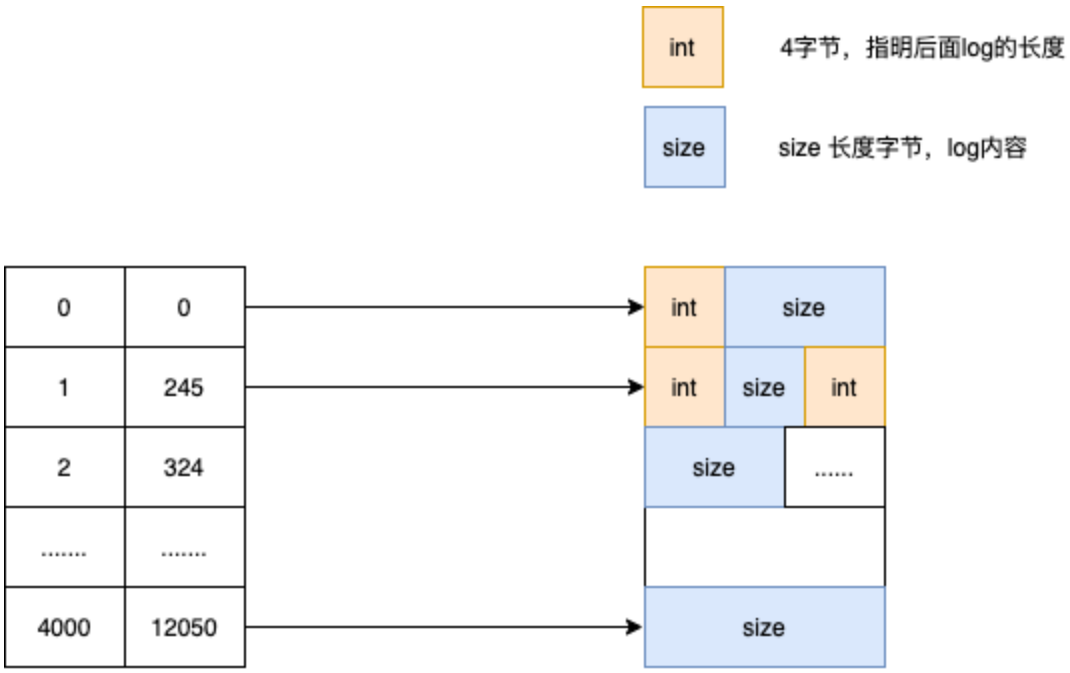
1. log index 内存中保留的形式
2. maxLogIndexSize内存中log index保留的最大数量。超过此数量就在logIndexMap中剔除log index较小的。可配置。假设在10000条的情况下，logIndexMap大约占用内存156KB

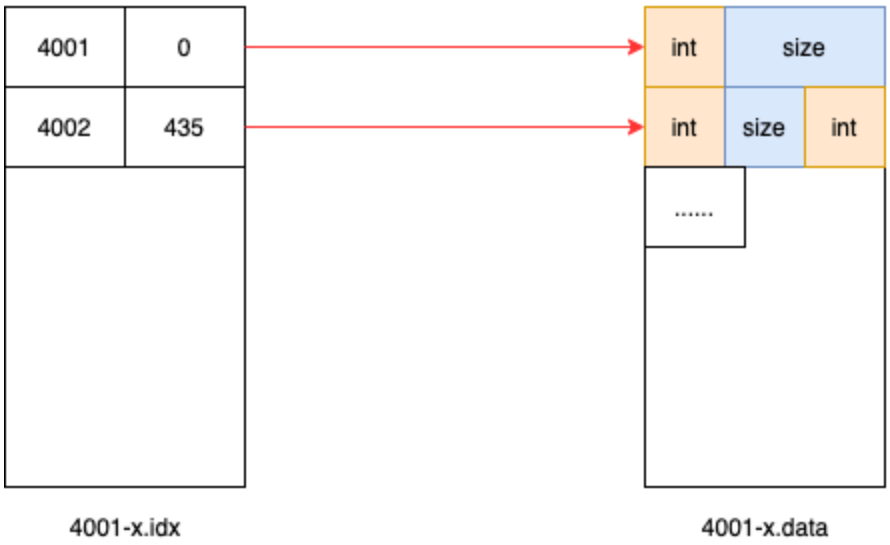
```
// log index
private Map<Long, Long> logIndexMap
// log indexlogIndexMaplog index
// 10000logIndexMap156KB
private int maxlogIndexSize
```

logIndexMap中 key 是log index， value是在log data 文件的offset。比如下图中<4002，435>则表明log index=4002的log日志，在log data 文件4001-X.data文件中。起始位置offset是435。Map有一个阈值限制(可配置)，用于控制内存中保留的最大的的log index索引数量。

磁盘上log
index文件也是多个<Long，Long>的组合。文件名是startIndex-endIndex.idx。当文件未关闭的时候，endIndex用X标识。关闭的时候替换为此时endIndex。

log data文件沿用目前设计，前4个字节表示log的长度，后面size字节表示log。





数据写入

每次新的请求到来会写入log data buffer和log index buffer，同时判断是否持久化。判断条件沿用现在的策略：当内存中log data buffer满了(默认开辟16MB)就进行刷写，刷写log data的同时刷写log index文件。

当log data文件大小超过1GB(可配置)，则新创建一个新的log data文件。同时切换新的log index文件，关闭旧的log data文件和log index文件。

log data 文件命名为 startIndex-endIndex.data。都是闭区间。

log index文件命名为startIndex-endIndex.idx。都是闭区间。

数据读取

提供如下接口

1. getLogs(long startIndex, long endIndex)：对外暴露的接口，给定startIndex和endIndex，都是闭区间，查询这两个Index之间的所有的日志，返回空，说明持久化的日志中也找不到所需的log，就需要发送snapshot。
2. getLogDataFileAndOffset(long startIndex, long endIndex);内部接口，查询index索引文件，获得[startIndex, endIndex]日志对应的文件名和offset。
3. getLogsFromOneLogDataFile(String fileName, Pair<startOffset, endOffset>);内部接口，针对某个文件和起始offset，查询对应的log data文件。

```
// startIndexendIndexIndex
// logsnapshot
public List<Log> getLogs(long startIndex, long endIndex);

// index[startIndex, endIndex] offset
private Map<String,Pair<startOffset, endOffset>>
getLogDataFileAndOffset(long startIndex, long endIndex);

// offset log data
private List<Log> getLogsFromOneLogDataFile(String fileName,
Pair<startOffset, endOffset>);
```

重启恢复

在本次开发的时候，也把maxAppliedIndex持久化到logMeta文件中。

重启的时候，读取出[maxAppliedIndex+1, maxCommitedIndex]的日志。重新进行apply。

重启不会主动在内存中构建logIndex索引，内存中logIndex索引只会在持续写入过程中更新，或是follower追赶leader的时候进行构建。目的是考虑到构建index也会有代价，而且重启的时候不一定被用到。

如果读取log index文件的时候发现文件损坏，尝试从此log index对应的log data构建索引，如果log data文件也损坏，则发送snapshot。

判断log data文件损坏规则：

1. 解析log data失败。
2. log data最后计算出来的log index与logMeta文件中持久化的不一致。

删除策略

采用条数限制和磁盘占用大小限制

1. log保留条数超过阈值。
    a. 假设保留 10min中的log，假设qps=1万，则10min中大约为600万条
    b. log 条数计算方式：通过log data 文件名即可计算。
2. log总的文件数量超过阈值：10。
    a. 以一个设备1000个测点为例，一批一行数据，则一个设备一批数据大约为1000*8=8KB。10GB数据大约为100万条log
    b. 数据大小计算方式：每个log data文件1GB左右，最多保留10个文件即可。
    c. 通过log data文件大小和最多保留的log data文件数量，即可控制log data占用的总的空间大小。

检查时机：单独起一个检查线程定时检查。1秒检查一次。

备注：删除是以文件为单位的。实际上磁盘占用的大小可能要比上述参数大。