分层存储集成测试工具开发

Created and last modified by 肖飞 on Dec 12, 2015

测试环境要求

- 测试最好在已测性能的drive上进行，这样的话需要更新drive性能配置文件为已测性能的drive，同时在测试机器上安装这样的drive。
- 测试也可以在没有测试性能的drive上进行，则默认drive的iops和bw带宽无限大，仅仅局限于容量，但是这样的话可能导致drive上IO过载，所以需要尽可能保证在iops和bw未知的情况下，IO不过载。
- 测试需要至少在包含两种不同介质磁盘的系统中进行。
- 集成测试和压力测试分开，在集成测试中可以只支持每种介质只一块drive的测试环境，如果在已测性能的drive上进行，则可以测试两种介质，如果在没有测试性能的drive上进行，则可以测试三种介质。

测试用例要求

- 创建的文件需要分布在不同介质上，在创建之后，能够获取该文件所在的介质，并记录文件和介质的映射关系（这可能需要在整个IO栈中添加相关的接口），以便在模拟重复IO时，可以选择某些特定的文件进行，这样在OSS端可以大概知道哪些文件应被向上迁移，哪些文件应该向下迁移，在AST端做出迁移决策之后，可以进行对比确定AST的效果，但是因为分层的决策时综合考虑响应时间和访问次数，所以OSS端的判断精确度有待商榷，这种方案相对也比较麻烦。另一种方案是，无论文件是如何在介质上分布的，对于所有的文件模拟出一定程度的随机IO，顺序IO，在OSS端测试程序中不做任何额外的统计，在AST端则将所有文件/对象的IO信息导出，将产生的advice信息导出，将产 生的plan信息导出，将migrate的对象信息导出，将migrated的对象信息导出，然后在AST中基于这些数据进行分析，查看结果是否符合预期。当前倾向于第二种方案。
- 测试需要运行较长时间，最好运行长达一个long term decision window + 一个short term decision window。
- 测试以BLOCK（BLOCK大小为4K）为单位进行，即无论是随机还是顺序IO，读写的offset都是BLOCK对齐的，len都是一个BLOCK的长度。
- 测试需要模拟以下情形：

  随机IO较多，顺序IO较多；

  随机IO较多，顺序IO较少；

  随机IO较少，顺序IO较多；

  随机IO较少，顺序IO较少；

- 测试可以进行定制，指定以下参数：

  读写模式（rw）：可选值为readonly，writeonly，readwrite，针对每一个文件，比如指定了readwrite模式，则在一个文件内部既会执行read，又会执行write，如果指定了readwrite模式，则还需要指定读写比例rwratio；

  随机度：又细分为顺序IO的文件比例（seqfileratio），随机IO 的文件比例（rndfileratio），混合IO的文件比例（mixfileratio），随机IO的随机跨度（rndstride），其中三中文件比例之和必须是100，如果指定了rndfileratio，还必须指定rndstride；

  块大小（blksize）：读写的单位，最小值为4K，最大值为64K；

  最大文件数目（maxfilenum）：最多这么多个文件参与测试；

  分层中每个cycle的时长（cycleperiod）：

  分层中short-term decision window包括多少个cycle（shorttermcycles）：

  分层中long-term decision window包括多少个cycle（longtermcycles）：

  测试运行多少个cycle（cyclecnt）：至少运行一个long term decision window + 一个short term decision window的时长，如果指定的cyclecnt小于一个long term decision window + 一个short term decision window，则将cyclecnt修正为一个long term decision window + 一个short term decision window。

  每个cycle执行多少个io（cycleionum）：每个cycle最多执行这么多的IO，这个值应该设置在可接受范围内，保证这么多io在一个cycle中肯定是可以完成的；在测试之初，可以通过测试完全随机IO（使用指定的随机跨度）的情况下能执行的IO数目对该参数进行调整。

  每个cycle执行多少个replay IO（cyclereplayionum）：每个cycle开始的时候首先执行这么多个replay IO，即对当前cycle之前已经执行过IO的文件再次执行IO，cyclereplayionum不应该超过cycleionum；

  是否将测试计划导出到文件（dumpplan）：如果将测试计划导出到文件，则以后的测试可以直接读取该文件来进行测试，如果指定了该选项，则必须同时设置dumpfile，指定导出文件；

  导出文件（dumpfile）：在指定了dumpplan的情况下必须指定；

- 为了满足后续需要进行的性能对比测试，所需要提供的支持

  后续需要测试在开启AST和不开启AST的情况下IO性能对比，所以要求两种测试环境下IO是可以重放的，也就是说无论开启AST与否，都执行同样的IO顺序，这要求后续测试中所谓的随机是一种可以预测的“伪随机”。

IO生成器

IO生成器负责调度产生下一个参与测试的文件inodeid，读还是写操作，IO操作的BLOCK index（BLOCK index从0开始，确定了BLOCK index就确定了offset和len）。实现以下功能：

- ywb_uin64_t GetNextInode（）

主要考虑replay（这里replay并不代表访问同一个BLOCK，而只是表示访问一个已经访问过的文件）与否，如果决定本次IO中要进行replay，则从已经执行过IO的inode中挑选，如果决定本次执行新的IO，则创建新的文件。

~~因为很难从外面指定的参数中确定接下来的IO中是否执行replay，所以该机制只能在代码内部制定，初步考虑方案为：每当一个新的cycle开始时，对过去已经执行过IO的所有的inode中“随机”挑选出m个inode进行replay（其中m通过以下方式计算：测试的时候如果获取一个新的inode，则该inode一定是当前系统中已经执行过IO的最大inode加1，所以当前已经执行过IO的inode数目就是等于当前最大inode，假设当前系统运行了ccnt个cycle，已经执行过的IO数目为iocnt，则m = iocnt/（2*ccnt），iocnt/ccnt表示每个cycle中执行IO的平均能力，除以2是希望匀出一部分IO能力给未执行过IO的文件）。注意这里的“随机”，是伪随机，至于怎么个伪随机法，会在后面阐述。~~

~~因为希望对inode上的IO区分出层次（对应于分层中的不同BUCKET），每次执行replay在挑选inode时采取如下方案：~~

~~每一个cycle都维护replay过的inode列表replayed，和新建的inode列表new；~~

~~第一个cycle执行完后，replayed列表为空，new列表不为空；~~

~~后续cycle逻辑如下：如果replayed列表不为空，则从replayed列表中挑选出一半（replayed列表中元素个数的一半）的inode，从new列表中挑选出另外一半的inode，将所有参与replay的inode添加到replayed列表中，并执行replay，并且对于该cycle中进行IO的所有新的inode都添加到new列表中。~~

~~这里提到在replayed列表中挑选，以及在new列表中挑选都是以下述方式进行：~~

~~挑选replayed列表中的索引为奇数的，即第1,3,5,7。。。个元素；~~

~~挑选new列表中索引为偶数的，即第2,4,6,8个元素；~~

针对杨苑和刘栋的改进意见，修改实现如下：

（1）先生成测试计划，测试计划是一个测试序列，后续执行测试的时候，依次执行该测试序列即可；

（2）在测试之前先进行初始化工作，其中一项工作是对所有参与IO的文件执行初始化，这里所谓的初始化即创建文件并写从第0个BLOCK到第（1M/blksize）个BLOCK；

（3）在分配新inode时，总是在当前未使用的最小inode；

（4）每当执行cycleionum个IO之后，总是从之前执行过replay的inode和上一个cycle中新的inode（记录在第0号replay IO列表中）中挑选出cyclereplayionum个inode执行replay IO，并将这cyclereplayionum个inode从之前的列表中移动到访问次数更高的列表中，另外cycleionum - cyclereplayionum个IO将在新的inode上进行，并将cycleionum - cyclereplayionum个inode放在第0号replay IO列表中；

（5）Cyclereplayionum个IO将从哪些inode中挑选呢？设计多个replay IO队列，每个队列存放执行过特定次数replay IO的inode，比如有m个队列，队列0存放上一个cycle中执行IO的所有新inode，队列1存放在截止当前cycle为止，所有执行过一次replay IO的inode，队列2存放截至目前为止，所有执行过两次replay IO的inode，以此类推。

队列个数等于longtermcycles + shorttermcycles。如果当前测试运行到第k * （longtermcycles + shorttermcycles ）+ 1次，则清空所有队列中的inode，并且该cycle 中全部在新的inode上执行IO，将这些inode全部存放到队列0上。每次从m个队列中选取cyclereplayionum个io时，如何在这m个队列上分配配额呢？如果cyclereplayionum小于m，则优先从索引较大的队列中选取，并记录最后一次获取inode的队列索引，下一次开始的时候从该索引开始朝索引较小的方向依次选取，直到索引为0.如果cyclereplayionum大于m，则依次从各队列中选取cyclereplayionum/m个inode，剩余的cyclereplayionum%m个inode，则采用类似于cyclereplayionum小于m情况下的处理方式来处理。至于在每一个队列上选取元素的方式，以队列index为种子来产生随机数，获取队列上被replay的inode。

（6）每当执行cycleionum个IO之后，就会执行cyclereplayionum个replay IO，以及cycleionum - cyclereplayionum个新IO。那么考虑到随机度参数，每个文件要么执行随机IO，要么执行顺序IO，要么执行混合IO，至于某个文件到底是执行哪种IO，都是在该文件第一次执行IO时根据随机度参数确定的（如配置选项中指定说要执行30%的随机IO，40%的顺序IO，30%的混合IO，也就是三种IO的比例为3:4:3，但是发现当前随机IO进行了3个，混合IO进行了3个，顺序IO只进行了2个，达不到指定的比例，那么接下来的新的inode上的IO一定是顺序IO，从此之后这个inode如果replay 也是按照第一次指定的随机模式进行）。系统中会分别为随机IO，顺序IO，混合IO维护自己的队列，在每次选择replay的时候，要从这三种队列中调度。所以对于（3）中在每种队列中调度的方法依然不变，只不过是要调度的inode个数不再是cyclereplayionum了。

（7）上述方案可能存在的问题

即使是完全一样的重放，并且在测试之初格式化硬盘也无法做到完全重放，因为硬盘分配哪个块也是不确定的。所以说只能是尽可能保证稳定的重放。同一种参数配置情 况下，如果要保证尽可能稳定，必须保证cycleionum是一个cycle中可以完成的，并且可以剩余一些时间空闲。

在每一个新的inode上的第一个测试都当做是顺序IO。

- ywb_uint32_t GetNextBlock（）

在确定在哪一个Inode上进行IO后就需要确定BLOCK了，这里直接根据该inode是从顺序IO的inode队列中获取的，还是随机IO的inode队列中获取的，还是混合IO的inode队列中获取的，确定其顺序性和随机性。因为当前OSS中顺序IO的判定是基于上一次访问的offset+len和当前访问的offset来确定的，所以在文件内部记录上一次访问的BLOCK index即可，如果要产生顺序IO，下一个IO的BLOCK index在前一个基础上加1即可，如果要产生随机IO，下一个IO的BLOCK index则为前一个基础上加rndstride即可。对于一个从未执行过IO的inode，第一次执行IO时一定从BLOCK 0开始。

- ywb_bool_t GetNextReadOrWrite（）

主要考虑单个文件内部的读写比例，在一个文件内部根据文件内部已经进行的IO进行决策；在任意时刻尽可能保证read和write的比例符合指定参数。

- ywb_status_t GenerateNextPlan（ywb_uint64_t* inode， ywb_uint32_t* block， ywb_bool_t* read， ywb_bool_t* replay）

该函数直接调用GetNextInode（）、GetNextBlock（）、GetNextReadOrWrite（）等函数。

- 对于新inode和replay inode的特殊处理

对于每一个新的inode的第一个操作一定是write，并且这第一个write操作不计算在read write比例的统计之中，因为对所有inode来说都是一样的。

- 如果没有新的inode可用情况下的特殊处理

~~如果在调度需要产生新的inode时候，没有可用的未曾执行过IO的inode，则重新从第一个inode开始，并且清空replayed列表和new列表。~~

如果在调度需要产生新的inode时候，没有可用的未曾执行过IO的inode，则只执行replay IO（即执行cycleionum个 replay IO），从此new列表就为空了，直到新的longtermcycles+shorttermcycles周期到来。

- 如果没有新的block可用情况下的特殊处理

直接从block 0开始，但是要增加一次随机次数。

- 每个cycle执行确定数目个io，并且要确定这么多io一定可以完成，而且还会剩余一部分io/bw带宽，这可以通过在指定测试配置之前测试获取