# 反向查询和column group的预读方案

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **编号** | **文档版本** | **修订章节** | **修订原因** | **修订日期** | **修订人** |
| **1** | 1.0 |  | 创建 | 2011.3.18 | 华庭 |
| **2** |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## 反向查询的预读

Advise()函数接收需要读取的所有block index info，这些block index info按照block存储在sstable中的顺序排序，如果反向查询，则反向遍历block info数组，决定当前aio buffer需要读取的block和预读buffer需要读取的block集合，从而实现反向预读。举个例子：

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

假设有10个block需要读取并且不考虑block cache存在某个block的情况，每个aio buffer只能存放2个block数据，当前的aio buffer为A，用于预读的aio buffer为B，如果为反向预读，A先读取block 9和block 10，B预读block 7和block 8，如果A中数据被应用层取完了，将B作为当前aio buffer，A作为预读aio buffer，并使用A预读block 5和block 6。依次循环，直到读取到所有的block数据。

如果为正向查询，则顺序读取block 1到block 10。对于正向查询或反向查询，传递给adivise()函数的参数都为需要读取的所有block index info数组和查询方向，正反向预读都由下层处理。

## 多column group的预读

当读取整行数据同时包含多个column group时，每个线程对每个column group都需要一个AIOBufferMgr的实例来读取column group的数据。我们以table包含2个column group为例，当应用层scan这个table中某个范围内的整行数据时，流程如下：

1. cs首先为column group A 创建sstable scanner A来读取行数据，为scan column group B创建sstable scanner B行数据。
2. 设置sstable scanner A的param，从线程私有实例中获得一个空闲AIOBufferMgr实例用于为sstable scanner A读取block数据，并调用adivise()告诉AIOBufferMgr需要读取的block集合，在advise()中将发起第一个AIOBuffer的数据读取。
3. 按照步骤2，对sstable scanner B做类似的预处理。
4. 如果sstable scanner A已经处理完成当前block或没有当前处理的block，则调用get\_block\_aio()从AIOBufferMgr中读取一个block，然后读取一行数据，再从sstable scanner B中读取一行数据，sstable scanner B同样会判断当前block是否已经处理完成或不存在，如果是则调用get\_block\_aio()从AIOBufferMgr中读取一个block。然后将sstable scanner A和sstable scanner B的行数据merge为table的一个整行。
5. 反复执行步骤4，每个sstable scanner的AIOBufferMgr都会根据当前AIOBuffer的状态发起预读，直到应用层读取到了所需的数据或ObScanner已经装满了，退出。

## copy2cache支持

advise()函数已经传入了所有需要读取的block info，每个block的info包括sstable\_id，offset和size，可以根据这些信息查询block cache，从而获得该block是否在block cache中存在。在advise()函数中会尝试分配当前AIOBuffer A和预读AIOBuffer B需要读取的block集合。处理步骤如下：

1. advise()函数顺序遍历需要读取block，查询block是否在block cache存在，如果存在并且A中还可以容下该block，将该block拷贝到A中。如果存在并且A中不能容下该block，尝试将该block加入B中。循环直到遇到第一个不在block cache中的block，将continue\_copy\_from\_blockcache标志设置为false，或者A和B都没有空间时，退出循环。
2. 检查A和B的状态，

1）如果A满了，B中block所占空间大于256KB，如果B也满了，将continue\_copy\_from\_blockcache标志设置为true，如果B中包含了最后一个block，标识数据已全部读取到AIOBuffer，退出advise()函数；

2）如果A满了，B中block所占空间小于256KB，如果B中包含了最后一个block，标识数据已全部读取到AIOBuffer，如果剩下需要从sstable文件中读取的数据量大于256K，扔掉B中的block数据，使用B发起预读，如果剩下需要从sstable文件中读取的数据量小于256K，仍然保留B中block数据；

3）如果A中有数据，B中没有数据，如果还有需要预读的数据，使用B发起预读；

4）如果A和B都没有数据，使用A读取数据。

1. Get\_block\_aio()函数依次从A和B中取出block数据。

1）如果A中包含有效block，返回该block。

2）如果A中block数据被完全取走。

如果continue\_copy\_from\_blockcache== true，尝试从block cache中拷贝数据到A

如果遇到不在block cache中的block，continue\_copy\_from\_blockcache=fasle。

如果A中的数据量小于256K，并且剩下需要从sstable文件中读取的数据量大于256K，扔掉A中的block数据，从sstable文件中预读block数据到A中。

如果剩下需要从sstable文件中读取的数据量小于256K，仍然保留A中block数据。

如果A中的数据量大于256K，保留A中block数据。

如果A中包含了最后一个block，标识数据已全部读取到AIOBuffer

如果continue\_copy\_from\_blockcache==false，直接从sstable文件中预读block数据到A中。

从B取出有效的block返回。

3）如果B中block数据也完全取走，重复类似2）中的步骤。

## 处理Scan提前退出

Scan因上层出错，或是ObScanner缓冲区已满，而AIO正在预读block数据，这些预读的数据将不再有效，但线程下次处理新的请求时，可能AIO读取还没有返回，这就会影响到下一次scan请求的响应时间，为了避免这种情况发生，在chunkserver的cs\_scan()函数调用send\_response()函数将响应数据写入packet的buffer以后，可以检查当前线程的AIOBuffer状态，如果有AIOBuffer处于WAIT状态，则等待该AIOBuffer返回，根据情况考虑是否将该数据拷贝到block cache中，如果所有的AIOBuffer都处于FREE状态或者等待超时了后，退出cs\_scan函数，可以处理下一个请求了。

当send\_response()函数将响应数据写入packet的buffer以后，并不确定tbnet的线程什么时候调用packet的encode()函数将packet的数据拷贝到tbnet的内部buffer并发生出去，但这个时候当前线程已经空闲了，可以检查AIO的状态了，并且不会影响到当前请求的响应时间。

每日合并是通过单独的线程处理，不会调用cs\_scan()，但每日合会顺序读取完整个文件，不存在丢弃无用数据的情况，不用特殊处理，即使在切换column group并且sstable分裂时，也不会预读到无效数据。

这种方案只处理cs\_scan()，也只有scan操作会用到AIO预读，所以当当前线程处理完一个scan请求后，可能下一次这个线程只是处理一个get请求，而get请求不需要AIO预读，是不用等待AIOBuffer空闲的，这种情况下就做了一些无用的等待工作。好在系统中大多为scan操作，这种影响可以忽略。

## 处理AIO读取长时间不返回

由于cs\_scan()已经等待了当前线程所有的AIOBuffer处于空闲状态或是等待超时，如果线程在试图从packet queue中获得一个packet进行处理时，发现线程的AIOBuffer不为FREE状态，意味着AIO读取出问题了，退出当前线程，如果多个线程都出错了，kill cs，别无选择。