## VHD格式文件在读写过程中涉及到的非对齐IO总结

### 写逻辑（以非differencing类型的dynamic类型的VHD为例）

**如果需要分配新的block**

1. 分配新的block，并将前一个block的尾部到新的block尾部的整个区域初始化为0（更新到VHD文件中）。新的block的数据区域（不包括bitmap部分）必须起始于页边界，且在前一个block和当前新的block之间的区域足够容纳bitmap，由于前一个block的数据区域（不包括bitmap部分）也必须起始于页边界，而每个block的大小一定是2的n次方个sector，只要n大于3，就一定能确保每个block是页对齐的，在默认情况下block大小为2M，每个block肯定是页对齐的，所以每个block的尾部也一定是页对齐的，前一个block的尾部和后一个block的数据区域之间预留m个页，这m个页刚好是bitmap区域大小向上对齐到页的结果。在分配新的block的时候，需要将上一个block的尾部到当前block的尾部的整个区域初始化为0，但是经过上面分析，可以确保是页对齐的。
2. 更新footer。因为footer只有512字节，所以会产生一个非对齐写IO。
3. 更新BAT。将BAT区域中该block对应的4字节所在的一整个sector从内存中读取出来，更新该block所在的那4个字节，然后更新整个sector的内容到VHD文件中。这里会产生非对齐写IO。
4. 写入数据。
5. 更新bitmap。将vhd\_bitmap::shadow（大小为bitmap区域向上对齐到sector）中的内容写入VHD文件中。这里可能产生非对齐写IO，取决于bitmap区域向上对齐到sector后实际占用的sector数目，以及bitmap区域起始sector。根据（1）中描述，在前一个block和后一个block之间预留了m个页的空间，用于存放bitmap，这m个页刚好是bitmap区域大小向上对齐到页的结果，对于默认的block大小来说，bitmap刚好占用一个sector的空间，所以m = 1，则这1个页的布局为：前7个sector是padding区，最后一个sector存放bitmap，所以更新bitmap的时候会产生非对齐写IO。

**如果不需要分配新的block，且tdbatmap区域中该block相应的bit置为1**

（1）直接更新数据即可。

**如果不需要分配新的block，且tdbatmap区域中该block相应的bit置为0**

1. 检查相应的bitmap是否在bitmap cache中。
2. 如果不在bitmap cache中，则参考“**如果不需要分配新的block，但是相应的bitmap不在bitmap cache中**”。
3. 如果在bitmap cache中，则参考“**如果不需要分配新的block，且bitmap在bitmap cache中，但是bitmap中相应的位是0**”或者“**如果不需要分配新的block，且bitmap在bitmap cache中，但是bitmap中相应的位是1**”。

**如果不需要分配新的block，但是相应的bitmap不在bitmap cache中**

1. 读取bitmap并添加到bitmap cache中，结合“**如果需要分配新的block**”中的分析，对于默认的block来说，一定会产生非对齐读IO。
2. 后续IO取决于bitmap中相应的bit是否被设置，参考“**如果不需要分配新的block，且bitmap在bitmap cache中，但是bitmap中相应的位是0**”和“**如果不需要分配新的block，且bitmap在bitmap cache中，但是bitmap中相应的位是1**”。

**如果不需要分配新的block，且bitmap在bitmap cache中，但是bitmap中相应的位是0**

1. 更新数据。
2. 更新bitmap。根据“**如果需要分配新的block**”中的分析，会产生非对齐写IO。

**如果不需要分配新的block，且bitmap在bitmap cache中，但是bitmap中相应的位是1**

1. 如果是非fixed类型的VHD，且更新的是除第一个block外的block中的第一个sector的数据，且batmap中记录该block是全满的，则写入带padding的bitmap，带padding的bitmap一定是页对齐的，不会产生非对齐IO。另外，本次写会写冗余的元数据，是对性能的一个优化（如果不做此优化，则会产生约25%的性能下降），根据schedule\_redundant\_bm\_write中的注释“This will help VHDs on raw block devices, while the FS-based VHDs shouldn't suffer much.”
2. 直接更新数据即可。

### 读逻辑（以非differencing类型的dynamic类型的VHD为例）

**如果相应的block不在BAT中（即相应的block不存在）**

1. 对于非differencing类型的dynamic的VHD，直接填充0返回。

**如果相应的block存在，且tdbatmap区域中该block相应的bit置为1**

（1）直接读取数据。

**如果相应的block存在，且tdbatmap区域中该block相应的bit置为0**

1. 检查相应的bitmap是否在bitmap cache中。
2. 如果不在bitmap cache中，则参考“**如果相应的block存在，但是相应的bitmap不在bitmap cache中**”。
3. 如果在bitmap cache中，后续IO取决于bitmap中相应的bit是否被设置，参考“**如果相应的block存在，且bitmap在bitmap cache中，但是bitmap中相应的位是0**”和“**如果相应的block存在，且bitmap在bitmap cache中，但是bitmap中相应的位是1**”。

**如果相应的block存在，但是相应的bitmap不在bitmap cache中**

1. 读取bitmap并添加到bitmap cache中，结合前面的分析，对于默认的block来说，一定会产生非对齐读IO。
2. 后续IO取决于bitmap中相应的bit是否被设置，参考“**如果相应的block存在，且bitmap在bitmap cache中，但是bitmap中相应的位是0**”和“**如果相应的block存在，且bitmap在bitmap cache中，但是bitmap中相应的位是1**”。

**如果相应的block存在，且bitmap在bitmap cache中，但是bitmap中相应的位是0**

（1）对于非differencing类型的dynamic的VHD，直接填充0返回。

**如果相应的block存在，且bitmap在bitmap cache中，但是bitmap中相应的位是1**

（1）直接读取数据。

## 关于bitmap cache

Bitmap cache中保存的每个bitmap都是关于某个block的完整的bitmap信息，所以对于连续访问某个block中的数据的情况，该bitmap cache是有效的，但是非常随机IO的情况下，该bitmap cache中的bitmap很容易失效。

Bitmap cache的大小为32，其中存放至多32个block的bitmap信息。

优化的重点是识别出bitmap所在的页，并将其尽可能及时的保存在OSS缓存中（以便在需要读取相关的bitmap时能够命中缓存）。