## 背景介绍

### 超融系统

典型的超融合系统如下：



### XenServer动态VHD

VHD包括3中类型：固定(Fixed)VHD，动态(Dynamic)VHD和差异化(Differencing)VHD。固定VHD在创建虚拟盘的时候直接分配虚拟盘大小的存储空间，当然还要加上一些相关的元数据；动态VHD是按需分配的，在任意时刻占用的存储空间大小都跟它实际使用的存储空间大小相同；差异化VHD则在parent VHD(可以是固定VHD，或者动态VHD，或者是差异化VHD)的基础上，只记录那些更新过的数据块来表示虚拟盘的当前状态。本文中所涉及的是动态VHD。

动态VHD的硬盘布局如下：



Hard Drive Footer中包含Magic，Original Size，Current Size，Checksum等信息。在动态VHD的尾部存储了Hard Drive Footer，在动态VHD的头部存储了Hard Drive Footer的镜像。每当分配一个新的Block的时候，VHD尾部的Hard Drive Footer都会被重新写入到新的尾部。

Sparse Drive Header中包含Max #BAT entries，Block Size，Checksum等信息。Max #BAT entries表示BAT区域中最多存储多少个Block的地址信息，其实就是该VHD中最大包含多少个Block；Block Size表示该VHD中每个Block的大小，动态VHD在扩展的过程中会分配新的Block。

BAT(Block Allocation Table)中记录所有分配的Blocks的起始Sector地址。BAT区域所需要的空间大小在创建虚拟盘的时候按照虚拟盘的最大容量和Block Size计算出来，比如一个2GB的虚拟盘，如果每个Block大小是2M，则BAT中包含1024个entry，每一个entry占用4 bytes，表示相应的Data Block的起始Sector地址，初始值为0xFFFFFFFF。比如BAT #123中记录的值为0x12345678，则表示虚拟磁盘中第123个Block的内容存放于VHD文件中以第0x12345678个Sector为起始地址的Block中。BAT区域所占用的空间大小一定会向上对齐到Sector边界。

Tdbatmap区域中每个bit位代表相应的Block是否全满，如果为1，则表明相应的block全满，否则表明相应的block中只有部分数据或者不包含任何数据。Tdbatmap区域所占用的空间大小一定会向上对齐到Sector边界。

每个Data Block包括2部分：Block Bitmap和Block Data。Block Bitmap中的每个bit用于表示Block Data中的相应Sector中是否包含有效数据，如果某个bit位为1，则表示相应的Sector中包含有效数据。Block Bitmap区域所占用的空间大小会向上对齐到Sector边界。Block Data区域的大小一定是2^n个Sector，默认是2M，也就是4096个Sector，那么Block Bitmap区域占用空间大小就是512 bytes，也就是1个Sector。

从虚拟盘的一个Sector到动态VHD文件中的Sector的映射关系如下：

BlockNumber = floor(RawSectorNumber / SectorsPerBlock)

SectorInBlock = RawSectorNumber % SectorsPerBlock

ActualSectorLocation = BAT[BlockNumber] + BlockBitmapSectorCount + SectorInBlock

## XenServer中动态VHD的数据访问

### XenServer中关于动态VHD的实现

为了性能考虑，XenServer中Block Data区域的起始地址一定是页对齐的，两个Block Data之间Block Bitmap所在的区域也一定是页对齐的，即使Block Bitmap所需要的空间小于1个页，这一整页的空间也不用作其它目的，这是很重要的，因为Bitmap访问和更新是相对频繁的，如果Bitmap不是页对齐的，则在更新Bitmap的时候可能会带来非对齐IO，给VHD的底层存储系统带来额外的负担(更新之前需要事先Read一整页的数据)。以Block Data区域大小为2MB为例，Data Block的布局如下：



XenServer为动态VHD中的Hard Drive Footer，Sparse Drive Header，BAT，Tdbatmap均维护了相应的内存结构，这些内存结构用于加速在访问动态VHD文件过程中的元数据访问，所有这些区域中的数据一旦发生变更，这些内存结构都会被更新。除了Tdbatmap之外，Hard Drive Footer，Sparse Drive Header，BAT中的数据一旦发生变更，就会立即更新到VHD对应的底层文件中，Tdbatmap区域的数据发生变更时，只更新它的内存结构，只在VHD close的时候才会更新VHD对应的底层文件，因为Tdbatmap主要用于加速元数据访问，它对数据一致性要求较低。

为了加速Block Bitmap读取，XenServer还维护了Bitmap Cache，Bitmap Cache大小为32，只能缓存32个Data Block的bitmap信息。Bitmap Cache在数据访问具有局部性的场景下比较有用，但是在服务器虚拟化场景下，可能存在大量随机IO，此时Bitmap Cache可能命中率很低，Bitmap Cache的作用就大打折扣。

### 动态VHD写流程



从动态VHD写流程可知，在写IO过程中，除了要更新数据自身以外，还可能涉及BAT，Tdbatmap，Bitmap Cache和Bitmap等的读取以及Hard Drive Footer，BAT，Tdbatmap和Block Bitmap等的更新。Bitmap的读取需要从VHD底层文件读取，Hard Drive Footer，BAT和Block Bitmap等的更新需要向VHD底层文件写入。Hard Drive Footer和BAT的更新可能带来非页对齐的IO。

### 动态VHD读流程



从动态VHD读流程可知，在读IO过程中，除了要读取数据自身以外，还可能涉及BAT，Tdbatmap，Bitmap Cache和Bitmap等的读取以及Tdbatmap，Bitmap Cache和Block Bitmap等的更新。Bitmap的读取需要从VHD底层文件读取，Block Bitmap的更新需要向VHD底层文件写入。

## XenServer动态VHD在超融合环境下的性能问题

每个虚拟机在执行IO的过程中，除了正常的数据IO，还可能会在VHD的底层文件系统上产生以下IO：关于Hard Drive Footer和BAT的非页对齐写IO，关于Block Bitmap的页对齐的写IO或者读IO。每当分配新的Block的时候都需要产生关于Hard Drive Footer和BAT的非页对齐的写IO，每当第一次写某个Sector的时候都需要产生关于Block Bitmap的页对齐的写IO，每当读取或者写入的时候如果Block Bitmap没有在Block Bitmap Cache中命中，都需要产生关于Block Bitmap的页对齐的读IO。

虚拟机在执行IO的过程中，如果数据访问不具有局部性，则每个IO都会产生一个关于Block Bitmap的页对齐的IO，且这些IO在VHD底层的共享分布式存储系统上表现为随机IO。

虚拟机在执行IO的过程中，即使每个虚拟机上的数据访问具有局部性，在超融合环境下，动辄数十甚至数百个虚拟机共享访问底层分布式存储系统，多个虚拟机的IO交织在一起，在VHD底层的共享分布式存储系统上也表现为随机IO。

VHD底层的共享分布式存储系统是基于HDD硬盘的，对于随机IO是非常不友好的。因此针对这些随机IO进行优化是非常有必要的。

## 优化方案

为了行文方便，将Hard Drive Footer，BAT和Block Bitmap统称为元数据。

### 优化前IO路径介绍

在优化之前，所有的IO都发生在HDD中，数据和元数据的读写路径如下：



所有IO，无论是数据IO还是元数据IO，均发生在HDDIOModule，随机IO访问是一个灾难。

### 优化方案介绍

为了优化随机的元数据写IO，引入了基于SSD的LogIOModule。因为SSD具有相对较好的随机写IO能力，因此将随机的元数据写IO，写入到SSD中，既保证了数据的持久化，又提供了较好的写性能。

为了优化随机的元数据读IO，引入了基于内存的ReadCacheIOModule。将元数据保存在ReadCacheIOModule中，可以带来极致的元数据访问性能。

优化后的方案，对于数据的读写请求并没有任何改变，对于元数据的写操作则会在成功写了LogIOModule之后，再写入到ReadCacheIOModule中，对于元数据的读操作则会直接从ReadCacheIOModule中读取。



对于LogIOModule，采用append only模式，并且在内存中维护索引，方便快速定位到元数据，对于索引信息则只存储在内存中，在重启的时候，通过LogIOModule中的数据进行重建。因为采用append only模式，每次元数据更新都采用copy-on-write的方式重新生成一份数据，旧的元数据就不再有效了，随着时间的推移，LogIOModule中会存在严重的碎片化问题，因此在后台专门启动了一个碎片整理线程进行碎片整理，回收可用的空间。

因为LogIOModule中存储的的数据是持久化的，即使在重启的时候，也可以通过LogIOModule中的数据重建ReadCacheIOModule。

### 元数据IO识别

对于VHD底层的共享分布式存储系统来说，需要识别出哪些IO是数据相关的，哪些IO是元数据相关的，才能对元数据IO进行上述针对性的优化。

从VHD硬盘布局可知，Sparse Drive Header中包含Max #BAT entries，Block Size等信息，Max #BAT entries表示BAT区域中最多存储多少个Block的地址信息，其实就是该VHD中最大包含多少个Blocks，Block Size表示该VHD中每个Block的大小，从这2个字段就可以计算出来该VHD的最大大小，进而计算出VHD的硬盘布局中每个元数据区域的起始地址和所占用的空间大小，也就是计算出了VHD硬盘布局中每个元数据区域的地址空间范围，当虚拟机的IO到达时，如果IO落在某个元数据区域的地址空间范围内，则认为这个IO就是元数据相关的IO。

### ReadCacheIOModule进一步优化

ReadCacheIOModule中保存的主要是各Block的Block Bitmap信息。根据前面的介绍，XenServer VHD中，即使Block Bitmap占用空间少于1个page，也会实际占用1个page，这是为了确保关于Block Bitmap的写IO是页对齐的，否则在VHD的底层共享分布式存储系统中会产生非页对齐IO，在执行写IO之前必须先读取一整页的数据，然后更新之，最后才执行写IO。以VHD默认的Block大小(2M)为例，Block Bitmap占用空间为512 bytes，在更新Block Bitmap时，写的是Block Bitmap所在的一整页的数据，根据XenServer VHD硬盘布局的讲解可知，这一整页的数据中，除了最后一个Sector中Block Bitmap所占用的512 bytes之外，都是无效的数据，在ReadCacheIOModule中可以只保留这512 bytes的数据，在读取Block Bitmap的时候，可以将Block Bitmap所在的Sector填充为ReadCacheIOModule中保存的数据，其它Sector则可以直接填充为0。

因为Block中的每个Sector在Block Bitmap中占用1个bit，也就是Block和Block Bitmap空间占用是4096:1的关系，如果在ReadCacheIOModule中保存所有VHDs的所有的Block Bitmaps，则1G内存只能保存4T大小的VHD的元数据信息，对于超融合环境下，动辄数十甚至数百个虚拟机，存储总容量可能高达上百T，这需要较大的内存为ReadCacheIOModule服务，必须减少ReadCacheIOModule所占用的内存空间。为此，可以有2种方案：压缩存储和ReadCacheIOModule空间回收。

压缩存储，就不多说了，这里着重说下ReadCacheIOModule空间回收。如果某个VHD文件的某个的Block Bitmap中所有的bit位都为1，则可以将该Block Bitmap从ReadCacheIOModule中删除，因为该Block Bitmap后续将不会再被访问到了(Block Bitmap中所有的bit位都为1，则Tdbatmap中会将该Block对应的bit位置为1，访问Tdbatmap时发现该Block对应的bit位为1，就不会再读取Block Bitmap了)。如果某个VHD文件中所有的Block Bitmap中所有的bit位都为1，则ReadCacheIOModule中将不再包含关于该VHD文件的元数据。