深度解读 | 超融合快照原理，这是一篇完整指南帖！

QI (home.php?mod=space&uid=163807) 2020-4-11 15:49 5420

本帖最后由 QI 于 2020-4-17 14:02 编辑

> hello，大家好，本文主要介绍超融合的快照技术以及使用场景，希望大家可以通过帖，帮助到更多的人更快地从零开始了解快照技术。

前言

深信服超融合的快照技术是从计算快照到存储快照的转变，从COW到ROW的转变。超融合的虚拟化计算层是基于KVM来做的，KVM采用了qcow2的磁盘格式，而qcow2所支持的快照方式就是COW写时复制。KVM在虚拟化计算层对qcow2打快照时，会将生成的快照空间一起保存在原有的qcow2文件中，在删除或恢复快照的时候调用底层文件系统的指令来清理qcow2文件中的快照空间。

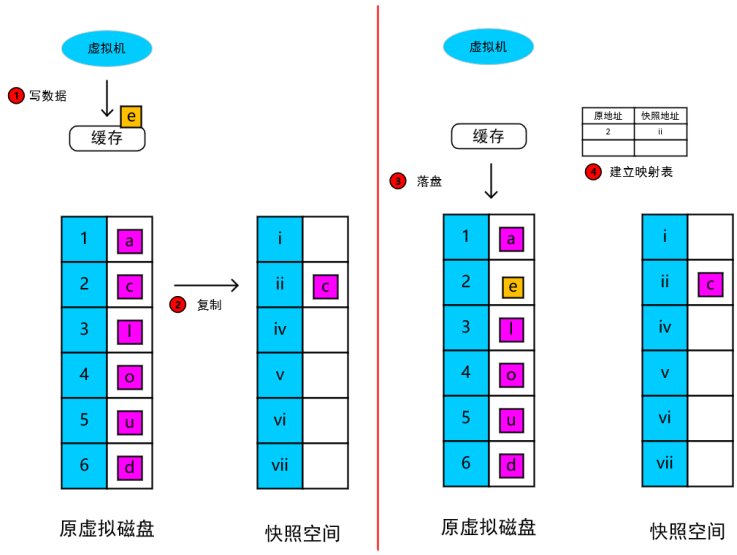
一开始超融合也沿用了KVM的COW，配合我们的aSAN虚拟存储来实现快照。这种方式就会有两个弊端，一个是因为COW快照技术本身带来的性能影响，在打完快照之后，数据写入时IO会被放大三倍，比较消耗存储的IO性能，这个细节原因会在后面讲COW原理的时候给大家讲到。另一个是因为我们的aSAN底层指令问题，在删除或恢复快照的时候无法清理qcow2文件中的快照空间。KVM被开发出来是基于Linux操作系统的，举个例子来说，如果底层使用的是Linux的ext3或ext4文件系统，那就可以通过调用它们的指令，来清理qcow2文件中的快照空间。但是呢在超融合中，用的是我们公司自己开发的aSAN虚拟存储，它底层的指令集不太一样，无法调用这个指令去清理快照空间。所以一直以来我们的aSAN虚拟存储上都无法去清理qcow2文件中的快照空间，只能是将快照的目录给删除掉。但是不要简单的以为其他厂商使用了KVM，就不能清理快照空间，这个主要还是看各自的底层文件系统是否支持。

为解决上述的两个问题现象，我们的aSAN开发了自己的存储快照（HCI 6.0.1），基于虚拟存储去做存储层面的快照，一方面存储快照选择使用ROW写时重定向技术来避免COW对性能的影响。另一方面从存储层面做快照可以使用aSAN文件系统自己的机制，解决qcow2文件中的快照文件残留问题，可以从底层去清理快照文件。这个存储快照的机制就是在打快照的时候，将生成的快照空间写成一份单独的qcow2文件，而不是把它放在原来的qcow2里，在需要清理这个快照空间的时候，只要把单独的qcow2快照文件删除掉就可以了。

不过大家要注意一下，由于我们的存储快照是基于aSAN虚拟存储去做的，所以当虚拟机存储在外置存储或本地存储的时候，是不支持存储快照的，只能用KVM的快照。另外两主机场景也是不支持存储快照及相关功能的。

接下来说一下COW和ROW的快照原理，在讨论快照原理的时候我们只讨论删改的那部分数据，因为对于新增数据来说，写在哪里都一样，不会对性能产生影响。

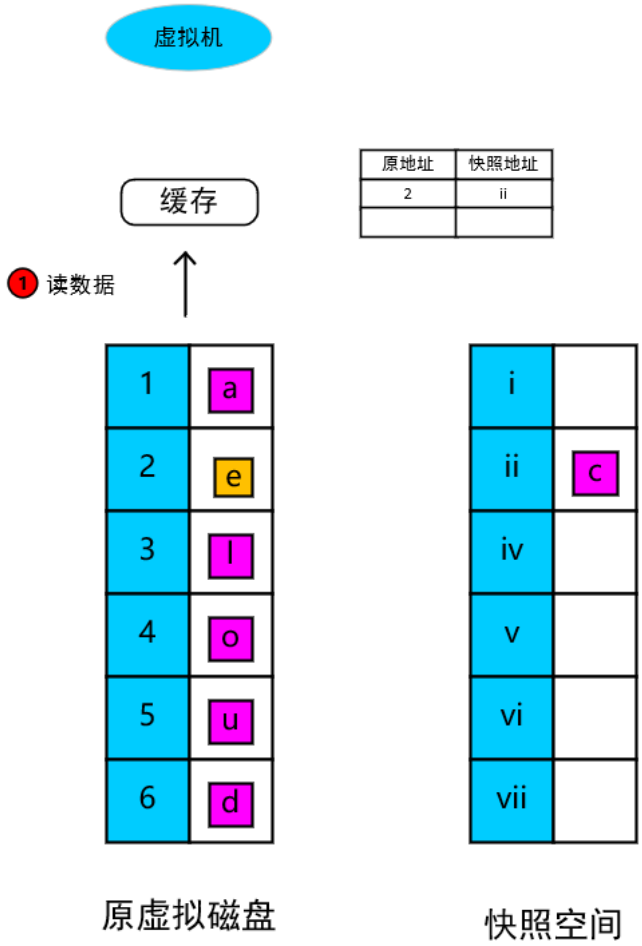
COW也叫写时复制，创建快照时，COW首先会为原虚拟磁盘创建一张数据指针表用于保存原虚拟磁盘的物理指针，也就是大家在图上看到左边蓝色框里的（1.2.3.4.5.6），然后创建（或者说划分）一个空的指针表，作为快照空间的数据指针表，就是图上右边蓝色部分的（i.ii.iv.v.vi.vii），并且会占用一部分存储空间, 用于保存快照后原虚拟磁盘中被更新的原数据。当要删改数据时，会先写入在缓存中等待，等待存储系统先将删改的原数据复制到快照空间里，然后再将缓存中的数据覆盖写到原虚拟磁盘上的位置，最后将原虚拟磁盘和快照空间的数据指针对应写入到一张映射表里。



COW后数据的写

为什么说COW写的时候会将IO放大三倍呢，图上蓝色的部分是数据指针，白色的部分是指针对应的存储空间，紫色和黄色表示数据块。此时虚拟机要将2地址的数据块c修改成e，e会先被写到缓存里，然后存储系统将原来的这个c复制到快照空间这里的ii位置，这是读了一次又写了一次，然后将缓存里的数据块e写入到原虚拟磁盘，又写了一次，一共是一次读和两次写，所以说IO会被放大。另外这个写入映射表的操作，其实只是写入地址指针，这个数据是非常小的，可以忽略掉。

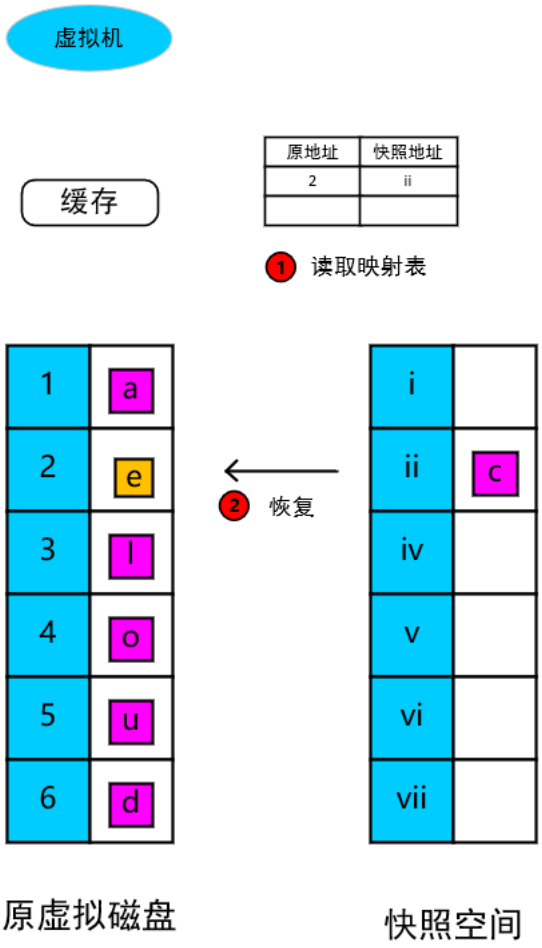
数据读取，COW快照创建后，虚拟机要读取当前最新的数据时，直接根据原虚拟磁盘的指针表读即可，无需去查询映射表和快照空间，因为修改的数据都已经保存在原虚拟磁盘里了。



COW之后数据的读

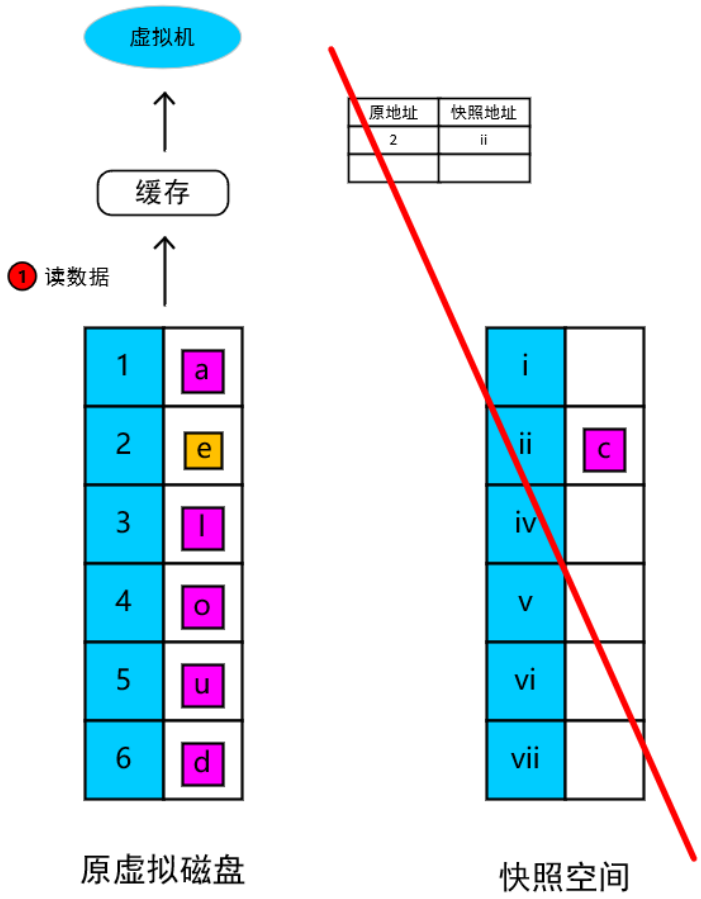
接下来讲快照的恢复和删除流程，这里大家注意，执行快照恢复后，虚拟机应该是打快照时的旧数据状态。而快照删除后，虚拟机应该是当前的新数据状态，这个先理解一下。

快照恢复是要回到打快照时的旧数据状态，那COW在恢复时，肯定是要将快照空间上的旧数据给回写到原虚拟磁盘上。它流程是这样的，首先要根据映射表，找到快照空间上的数据在原虚拟磁盘上所对应的位置，然后将数据回写到原虚拟磁盘上，覆盖掉修改后的数据。每个回写动作进行一次数据读和一次数据写，在数据全部回写完成之后，再将快照空间以及映射表删除掉。



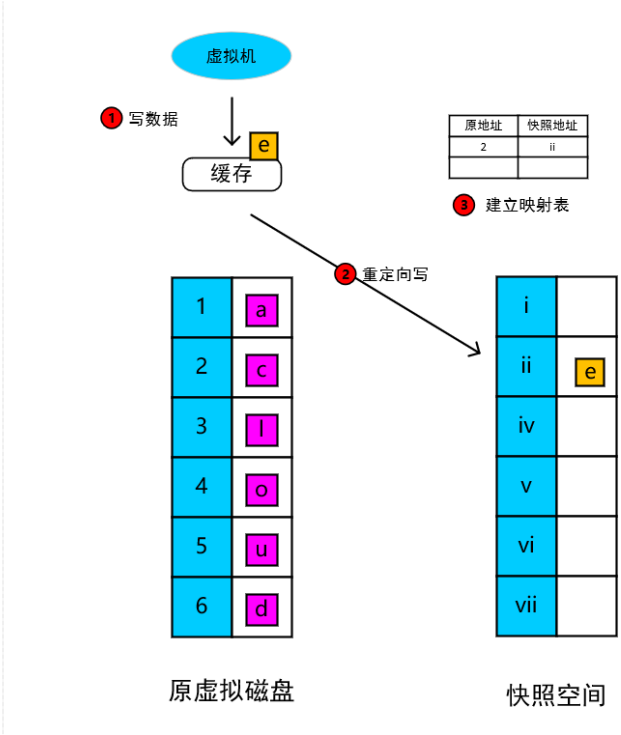
COW快照恢复

快照删除是要保持虚拟机最新的状态，那COW的删除就比较简单了，直接删除掉快照空间和映射表，就可以保证当前虚拟机在读原虚拟磁盘时是最新数据的状态，当然旧数据也会一起删除。



COW快照的删除

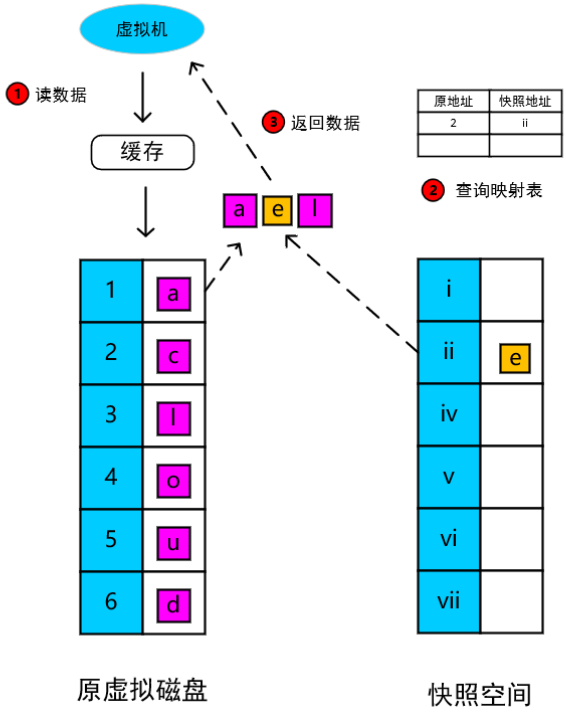
ROW也就是写时重定向，创建快照时，ROW首先会为原虚拟磁盘创建一张数据指针表用于保存原虚拟磁盘的物理指针,然后创建一块空的指针表，作为快照空间的数据指针表。同样的它会占用一部分存储空间, 用于保存快照后更新的数据。在我们的aSAN上这部分的存储空间是动态分配的，并没有一个起始的默认大小，但是最大不会超过虚拟机配置的磁盘大小。当要对原虚拟磁盘的数据进行删改时，会将删改后的数据写入到快照空间里，然后将原虚拟磁盘的地址指针和快照空间的地址对应写入到映射表里。



ROW后数据的写

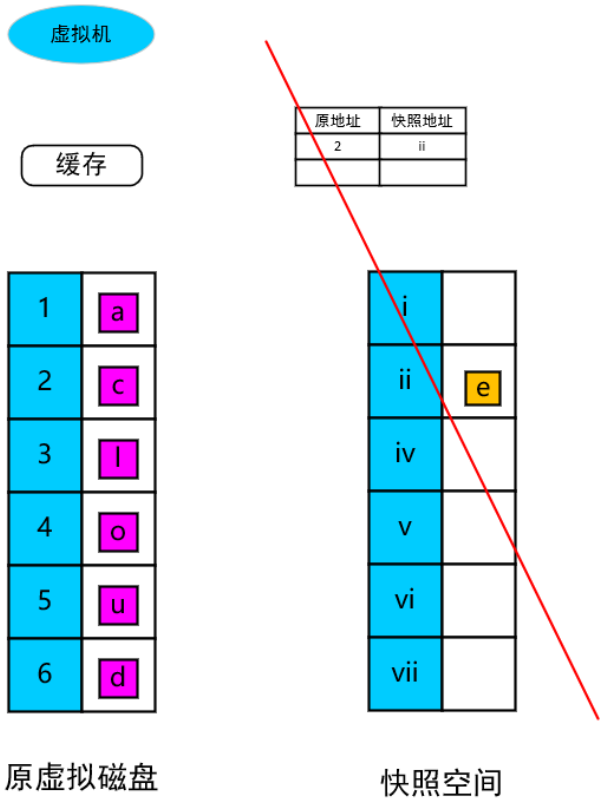
还是这个图，左边表示原虚拟磁盘，右边表示快照空间，蓝色表示指针表，白色表示存储空间，紫色和黄色的字母表示数据块。此时虚拟机要将2位置的c改写为e，首先会将e写入到缓存里，然后直接写入到快照空间的某块存储空间上（也就是右边这个e的位置），它不会像图上这样对起来，它也是在快照空间顺序写的，最后将这两个地址对应的写入到映射表里。可以看到ROW在写入的时候，是只有一次写入操作的。只是进行了写映射表的操作，这个前面也说了，记录地址的操作影响非常小，可以忽略掉。

ROW的读分为两种情况，一种是新增数据的读，新增数据的读直接从快照空间读取就好了，这个不会涉及到性能影响。另一种是删改数据的读，读删改数据的时候首先会查询映射表，找到对应快照空间的地址，再读到快照空间里的新数据。整体流程连起来大家可以看下：虚拟机想要顺序读原虚拟磁盘上的1到6位置上的数据块，读a和l时还是会从原虚拟磁盘读取，当读到2位置的c时，会通过查询映射表找到快照空间里的这个数据块e，返回给虚拟机。这里ROW读数据的时候有一个查询映射表的过程，是对地址指针的查询，比起指针写入的影响会大一点点，但是比起数据块的操作影响小很多，我们暂且认为它是有轻微影响。



ROW后数据的读

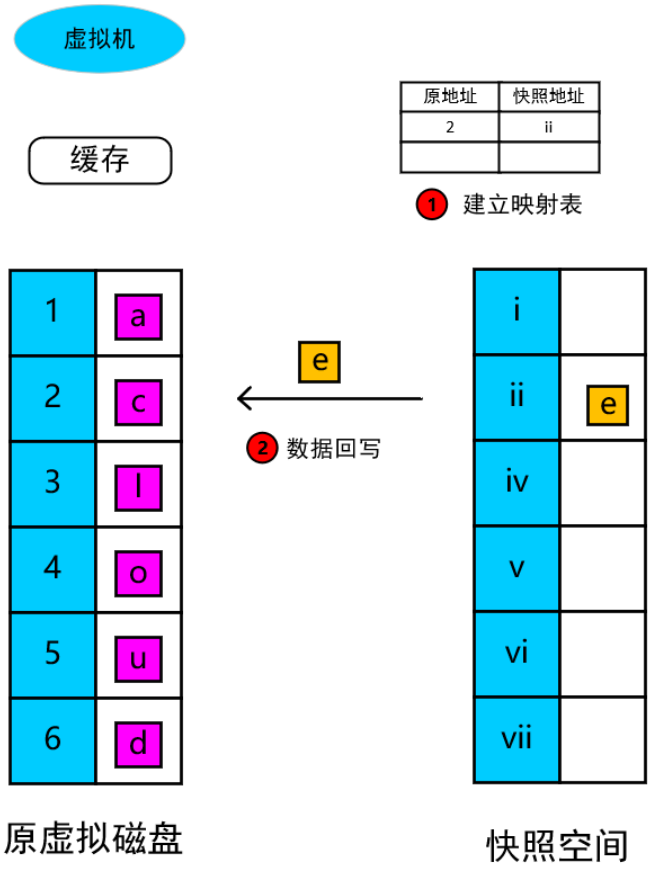
还记得我们说的快照恢复和删除的区别吗？快照恢复是要回到打快照时的旧数据状态，ROW的旧数据都保存在原虚拟磁盘里，所以在快照恢复的时候，直接删除掉快照空间和映射表就可以了。



ROW快照恢复

PS：说到快照恢复，为什么快照恢复的时候要关闭虚拟机？虚拟机运行过程中，内存里的数据是会持续往磁盘中写的，而快照恢复过程中是对磁盘有做回滚操作的，不可能做到在新数据写入的同时还能回滚旧数据。举个直观的例子，我现在有一台虚拟机刚安装好系统，打了个快照，然后我安装了微信、MOA软件，我正常使用这些软件的时候，我的电脑突然中毒了，此时我想对虚拟机恢复快照，如果不关闭虚拟机，这些软件一直在向磁盘写数据占用磁盘上的那个数据块。而恢复快照是要将这些软件整个覆盖擦除掉，这时候就会发生冲突。现在的快照技术在恢复时都是会将虚拟机重启的，不然就达不到快照恢复的目的。

快照删除是要保持虚拟机最新的状态，对于ROW来说，最新的数据是在快照空间中的，那在删除快照的时候，则需要依据映射表将快照空间的数据进行回写，再删除快照空间。大家可以发现ROW的恢复和删除过程与COW刚好相反，主要还是因为COW的原虚拟磁盘记录的是新的数据状态，而ROW的原虚拟磁盘记录的是旧的数据状态。



ROW快照删除

对比

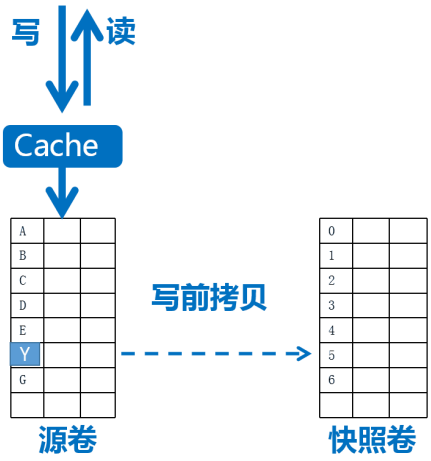
快照方式	COW	ROW
快照后写数据	受影响（3次操作）	不受影响（1次操作）
快照后读数据	不受影响（1次操作）	轻微影响（查询映射表）
快照恢复	需要数据回写	瞬间完成
快照删除	瞬间完成	需要数据回写
使用场景	读密集型	写密集型
其它	比较消耗IO性能	比较消耗计算资源

COW与ROW快照方式对比

这里对两种快照方式做了一个对比，快照后写操作时COW会进行三次数据操作，分别是旧数据读，旧数据写和新数据写，所以COW打完快照之后，写数据是会受影响的。而ROW读操作时会有轻微的影响主要是因为对地址映射表的查询动作。快照恢复和删除就不讲了，前面也都说过了，从场景来看快照的恢复场景肯定是比删除场景要多的，打快照当然是为了恢复的，不会有人没事儿打完快照删着玩儿吧，从这个角度来看ROW方式还是有优势的。但是当你要用到我们的定时快照功能时，这个快照删除还是会有一些影响的，在定时快照功能中，超出保存时间的快照会被删除，这个删除会产生一次读和一次写。从整体来看在ROW的技术背景下，即使使用了定时快照功能，性能影响也会比COW小很多。毕竟COW的每次写入都会被放大。

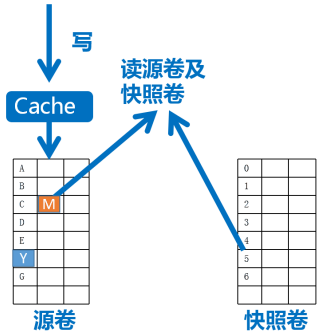
由于COW打完快照之后，数据的写入会受影响，所以COW比较适用于读密集型应用，比如说一些前端的WEB类应用，另外COW引起的数据读写的增加，主要消耗的是存储的IO性能。对于ROW来说，打完快照之后，数据的读取会受一些轻微影响，所以ROW比较适用于数据库类的写密集型应用，反复查询映射表主要消耗的是虚拟机的计算资源。

最后给大家放两张动图，体会一下COW快照后的读，和ROW快照后读写



COW后数据写（读直接读源卷）

ROW(Redirect-on-write)，也称为写时重定向。把对数据卷的写请求重定向给了快照预留的存储空间，直接将数据写入快照卷。读时，创建快照前的数据从源卷读，创建快照后的，从快照卷读



ROW后数据写与读