阿里巴巴云计算 - *Aliyun*

为分布式存储系统使用Bcache做本地缓存

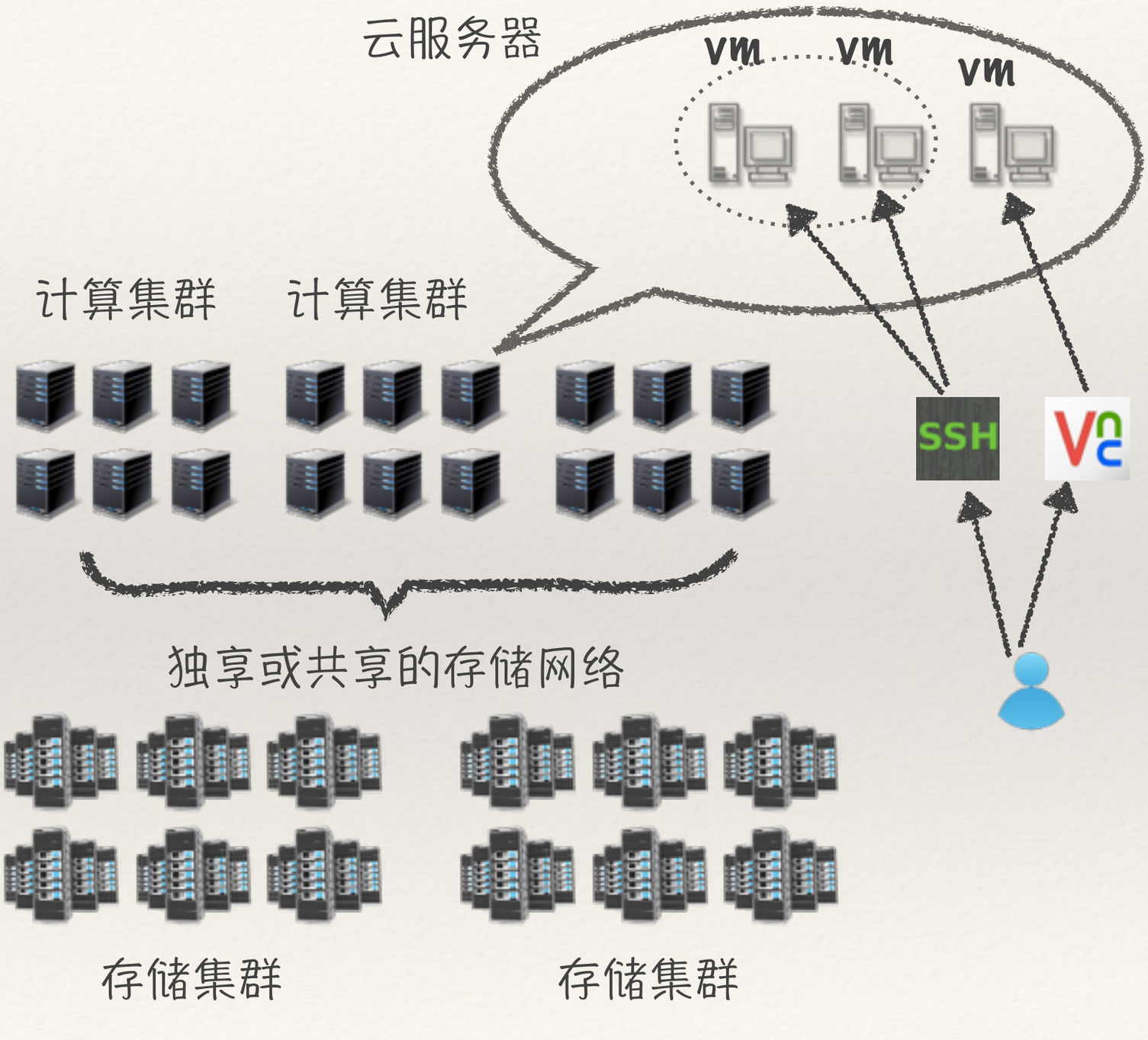
朱延海
gaoyang.zyh@taobao.com

目录

- 阿里云的存储系统
- 使用客户端缓存的收益
- 为什么选中Bcache
- Bcache的设计概述
- 我们做了哪些工作
- 未来的计划

阿里云的存储系统

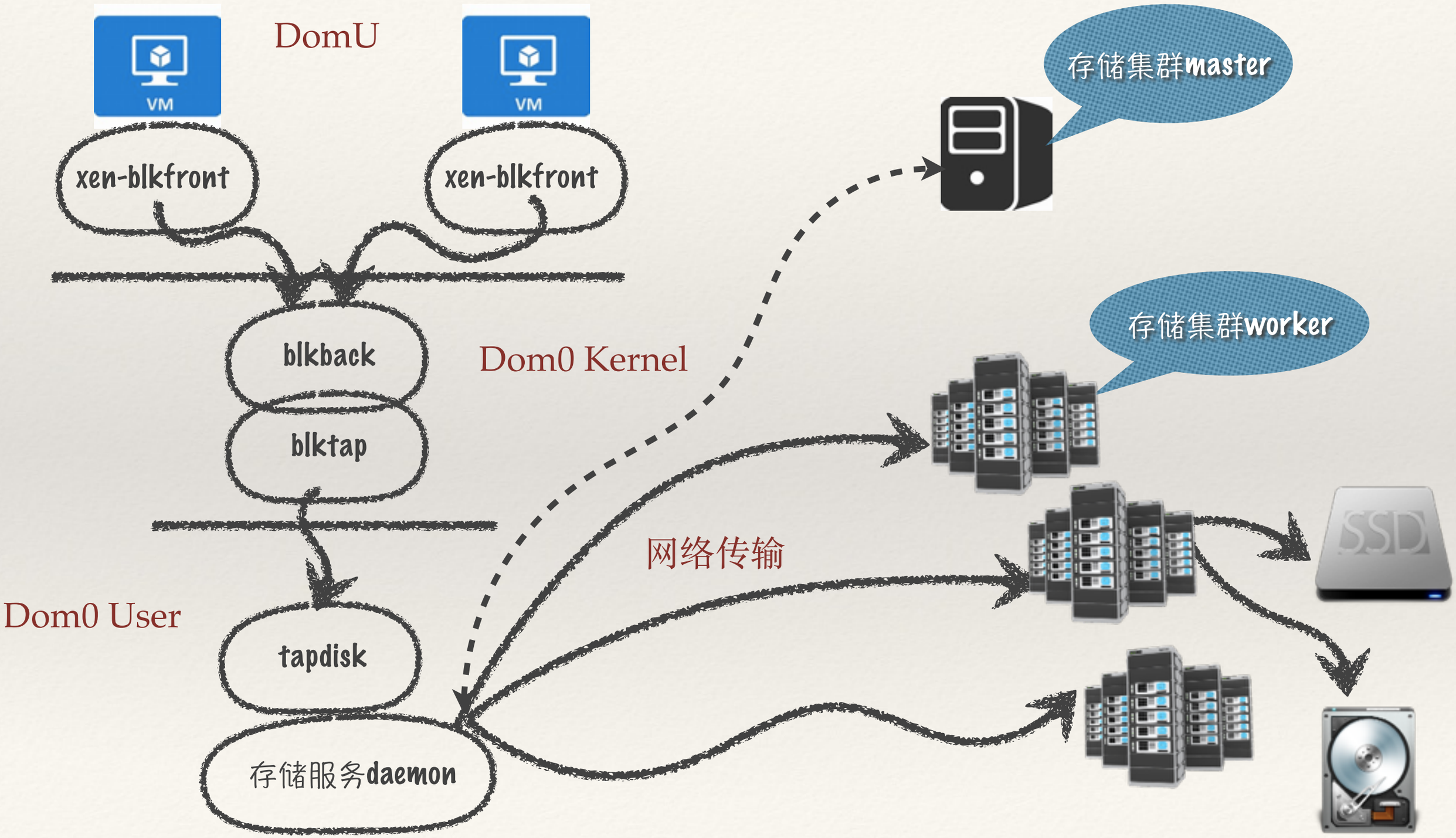
- 计算集群与存储集群物理上可能是同一个集群，也可以是两个不同集群
- 数据以三副本的形式强一致存储在chunk server上
- 千兆网和万兆网并存
- 新机器和老机器并存
  - IaaS的热迁移
- 服务的客户千差万别



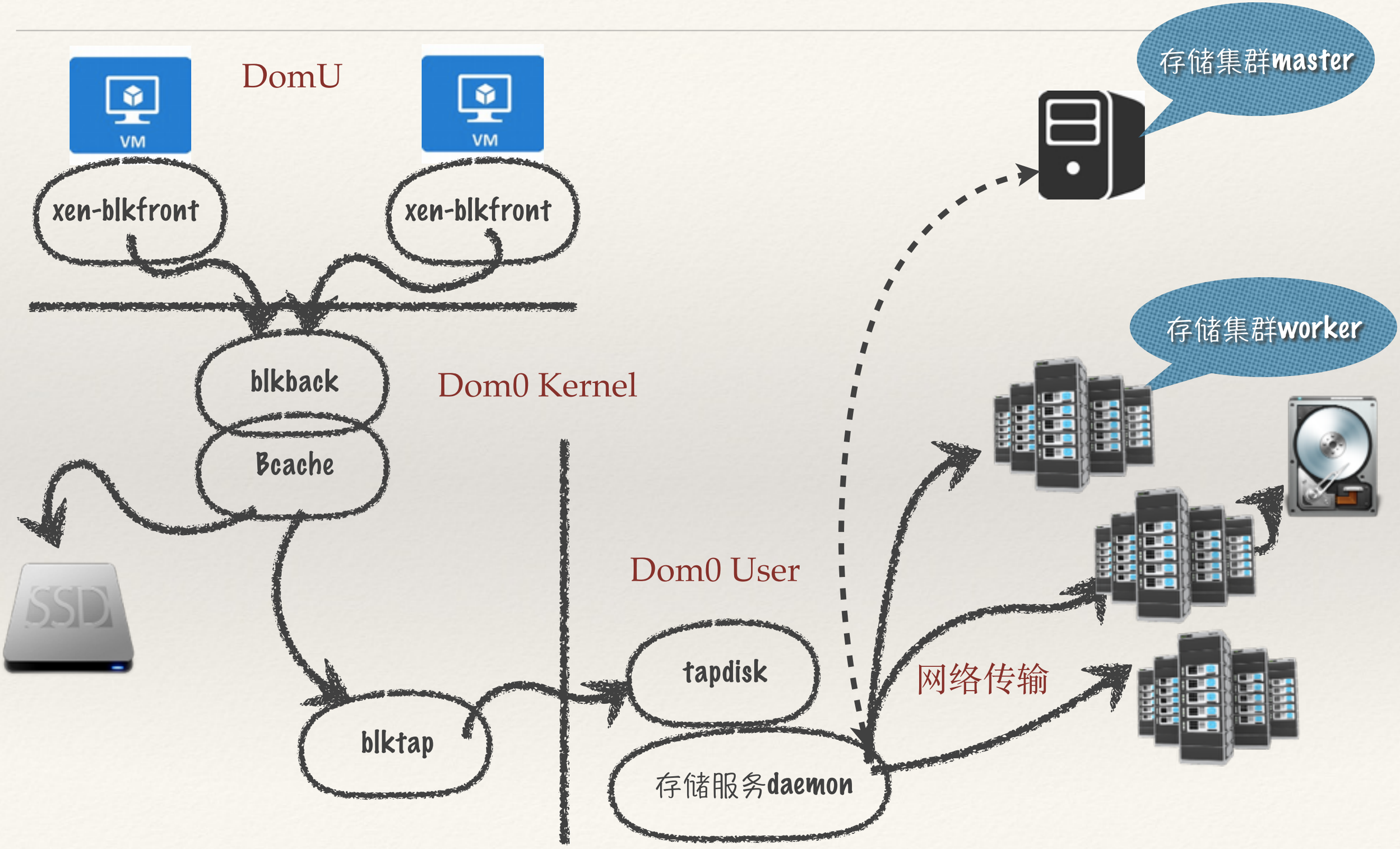
客户端缓存的收益

- 分布式存储的物理特性与本地磁盘不同
  - 支持的并发度非常高
  - 对IO模式是随机的还是顺序的不敏感
    - sata硬盘：随机访问3ms，顺序访问600us
    - 延迟长尾比硬盘更明显，经由网络的传输不可控因素更多
    - 与虚拟机的普通网络访问竞争带宽
- 用户对云存储的“期待”来自于传统磁盘的体验，使用客户端缓存
  - 节省网络流量，减少服务端资源占用
  - 向传统应用屏蔽网络存储特性
  - 构造短小的数据平面，更容易利用诸如NVMe等现代高速设备

在哪一层做缓存?


DomU
VM
VM
xen-blkfront
xen-blkfront
blkback
blktap
tapdisk
存储服务daemon
Dom0 Kernel
Dom0 User
网络传输
存储集群master
存储集群worker
SSD

在哪一层做缓存?


DomU
VM
VM
xen-blkfront
xen-blkfront
blkback
Bcache
Dom0 Kernel
Dom0 User
SSD
blktap
tapdisk
存储服务daemon
网络传输
存储集群master
存储集群worker

Flashcache? pblcache?...

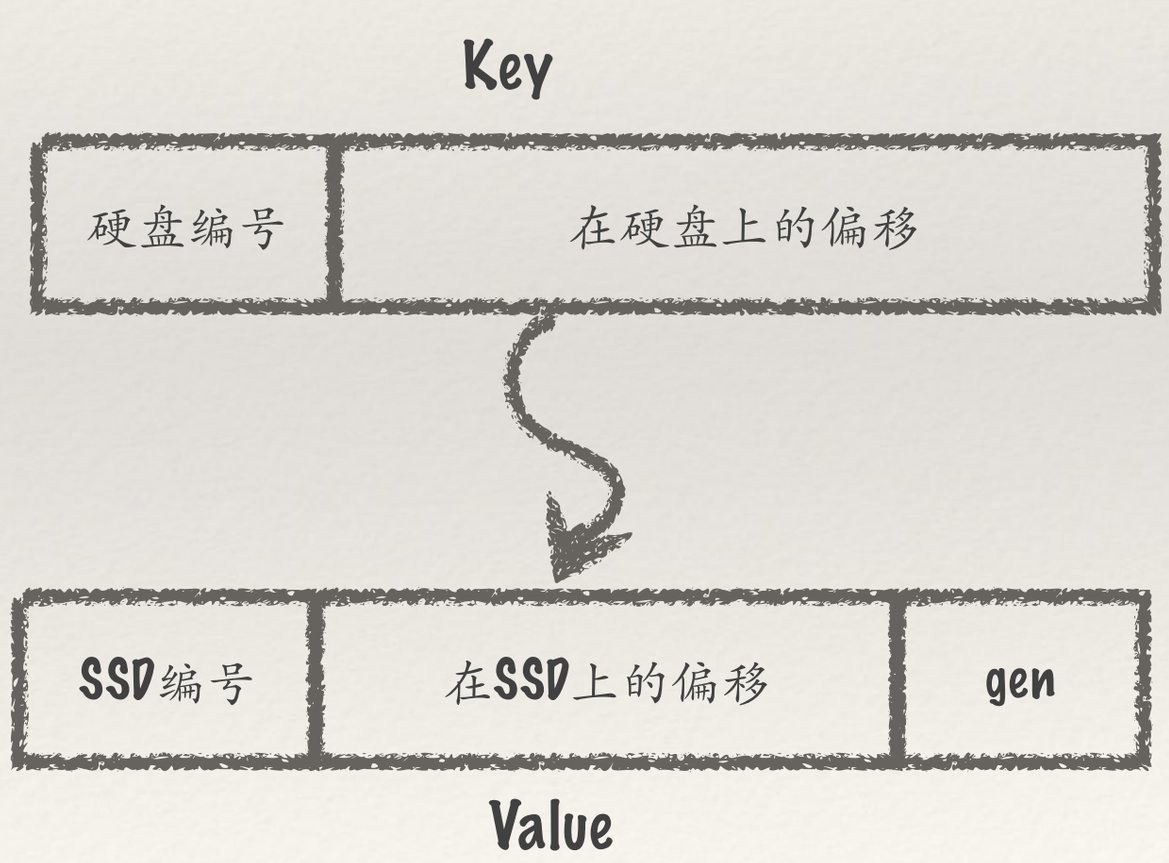
- Flashcache严重依赖于一个重要假定：在它的使用场景下只需要缓存随机IO
  - 基于组相联Hash，在各个2MB组内淘汰
  - 桶内淘汰，一次顺序写会冲掉其他Hash冲突的数据
    - 原因：对于随机请求，有多个较热的块集中在一个桶里的概率很低
    - 在支持一对多加速后此问题更严重
  - 解决方案
    - 检测到顺序读写后直接跳过SSD
    - Random Hash，故意把顺序IO变为随机IO
- dm-cache
- pblcache
  - 一个独立运行的daemon，qemu通过ipc与其通信

Bcache的基本情况

- 一个缓存池由一个或多个SSD组成，一个系统中可以有多个缓存池存在，每个缓存池可以为多个后端设备服务。
- 每个连接到缓存池的后端设备对应一个虚拟块设备
- 由一个准B+树索引。具有Cow式的分配器，mark-and-sweep GC。Bcache现有代码上再加一点点，就可以实现一个文件系统
  - Update: “[ANNOUNCE] BcacheFS” - https://lkml.org/lkml/2015/8/21/22
- 默认分配单位512KB
- 淘汰算法：LRU、FIFO、随机
- 缓存模式：writeback、writethrough、writearound
- CoW Btree，数据结点不会在原地修改，元数据结点视情况而定
- 使用Journal加速元数据写入
- 各种小feature，例如SSD出错时尽量试图令其不影响用户IO、SSD拥塞时自动访问后端设备等等
- 3.10进入主线内核，但最近更新缓慢，邮件列表也不活跃。

检索和分配

- 将后端设备序号统一编码进一个平坦地址空间
- 以后端设备偏移为索引，指向其在SSD上的偏移
- struct bkey {
  - uint64_t high;
  - uint64_t low;
  - uint64_t ptr[];
- }



检索和分配

- 从存储系统的各个标准设计元素中挑出想要的……

1. 设想一个标准的B+树
2. 随机的插入和删除B+树中的key会带来大量随机IO
   - 加入journal (i.e. redo log)
   - B+树结点的修改延迟落盘(30s)
   - 遇到BIO_FLUSH/BIO_FUA时等待结点修改落盘
3. 数据的随机写带来SSD上的随机IO和partial write
   - Cow 不在原地修改
   - Mark-and-sweep GC
4. 常态运行时ssd基本是满的，元数据查找开销较大
   - 让树结构尽可能扁平，默认结点大小512KB
5. 在一个512KB结点内的有序key序列上查找很慢。
   - 把一个大结点拆成若干小结点(bset)，并且小结点在内存中的形态与磁盘上不同 （类似LSM）

淘汰

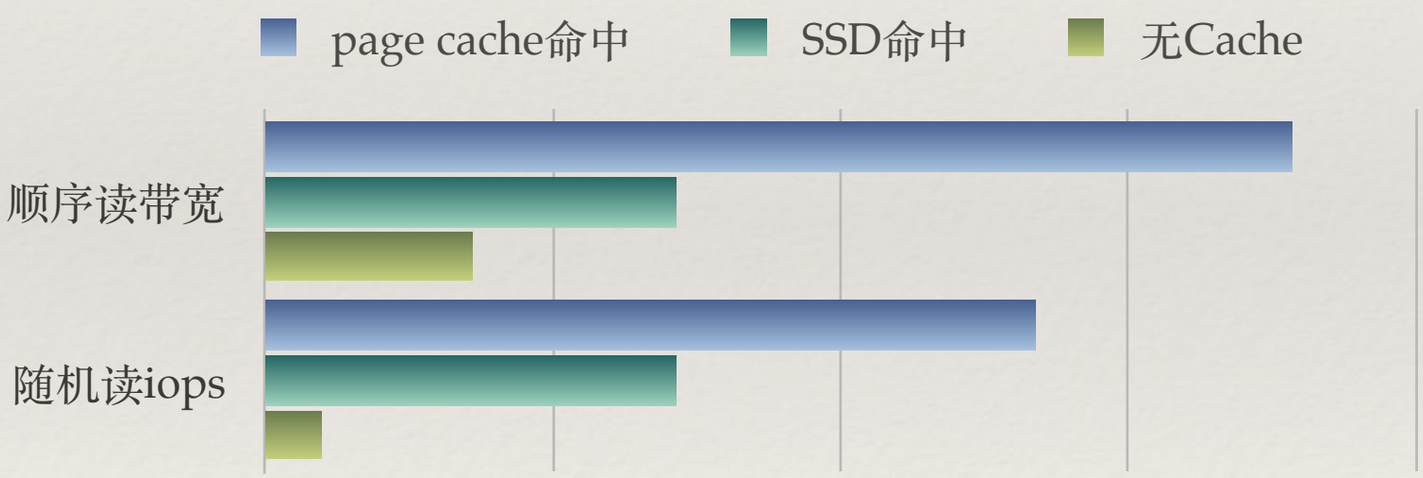
- 运行一个mark-and-sweep式的异步GC过程
- 每个512KB桶有一个gen，以512KB为单位淘汰，淘汰时桶的gen++
  - 每个指针上记录着自己“期待”的指向的数据桶的gen
    - 单独维护一个桶到gen的查找表格
    - 桶的gen的改变会持久化，计入journal
- 检索数据时发现指针上的gen < 目标桶的gen，说明该指针已无效；无效的指针由gc异步删除掉
  - gc线程的功能：compaction + 淘汰 + lazy gc
- 目前只实现了LRU、FIFO、随机三种淘汰

产品化(1)

- 开源项目的“Production Ready” 需要大量琐碎的细节工作
- Backport
  - 4.2 -> patched 2.6.32
- 在线热升级、降级
  - blkback需要支持动态更换下边连接的设备
  - Bcache有大量64KB的连续物理内存分配，机器跑了比较久之后很难insmod
  - 热升级失败之后的回滚
- 延迟长尾
  - 在writeback和GC线程活动持有一些重要读写锁的写锁，会明显造成IO操作延迟长尾
  - 检测持锁时间，将critial section分段
- 关机
  - 无法区分出这次关机是重启、暂时关机、打算转移到别的宿主机或可用区上，还是永久的销毁
  - 只能让关机等待脏数据刷干净，造成关机非常慢

产品化(2)

- 在Generic Block Layer之下利用Page Cache建立一级内存缓存。获得一个内存+SSD的两级缓存
  - 幸运的内存命中可以有效提升iops、降低延迟
  - SSD在读写混合时性能最差，回写脏数据时从page cache中读出可减轻对用户IO的影响



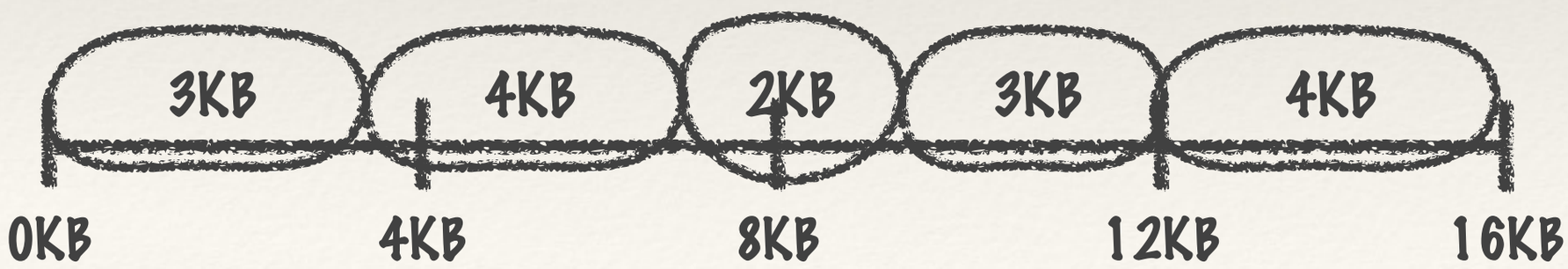
* 使用fio测试, 顺序读bs=1MB, iodepth=128, 随机读bs=4KB, iodepth=128

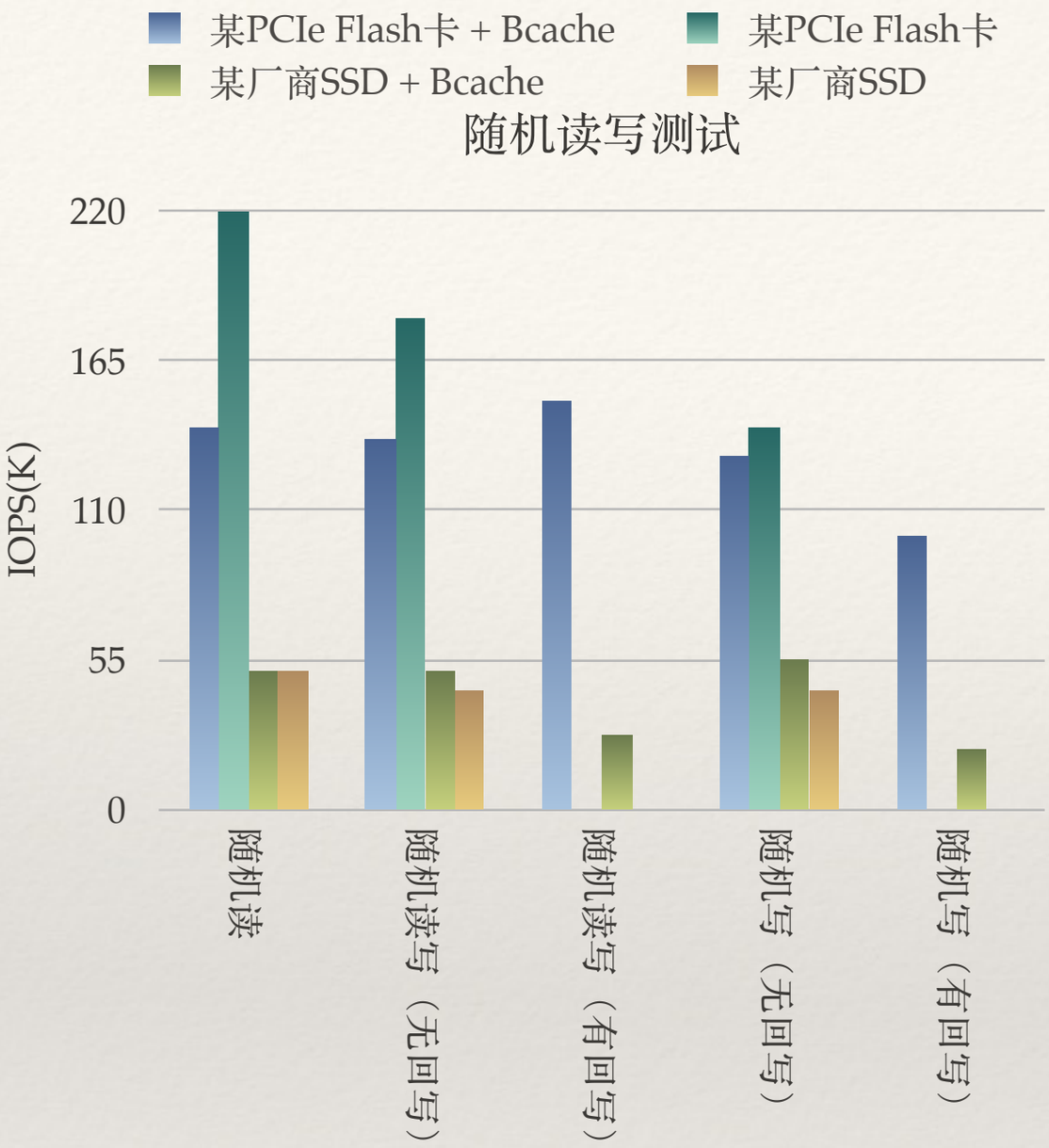
产品化(3)

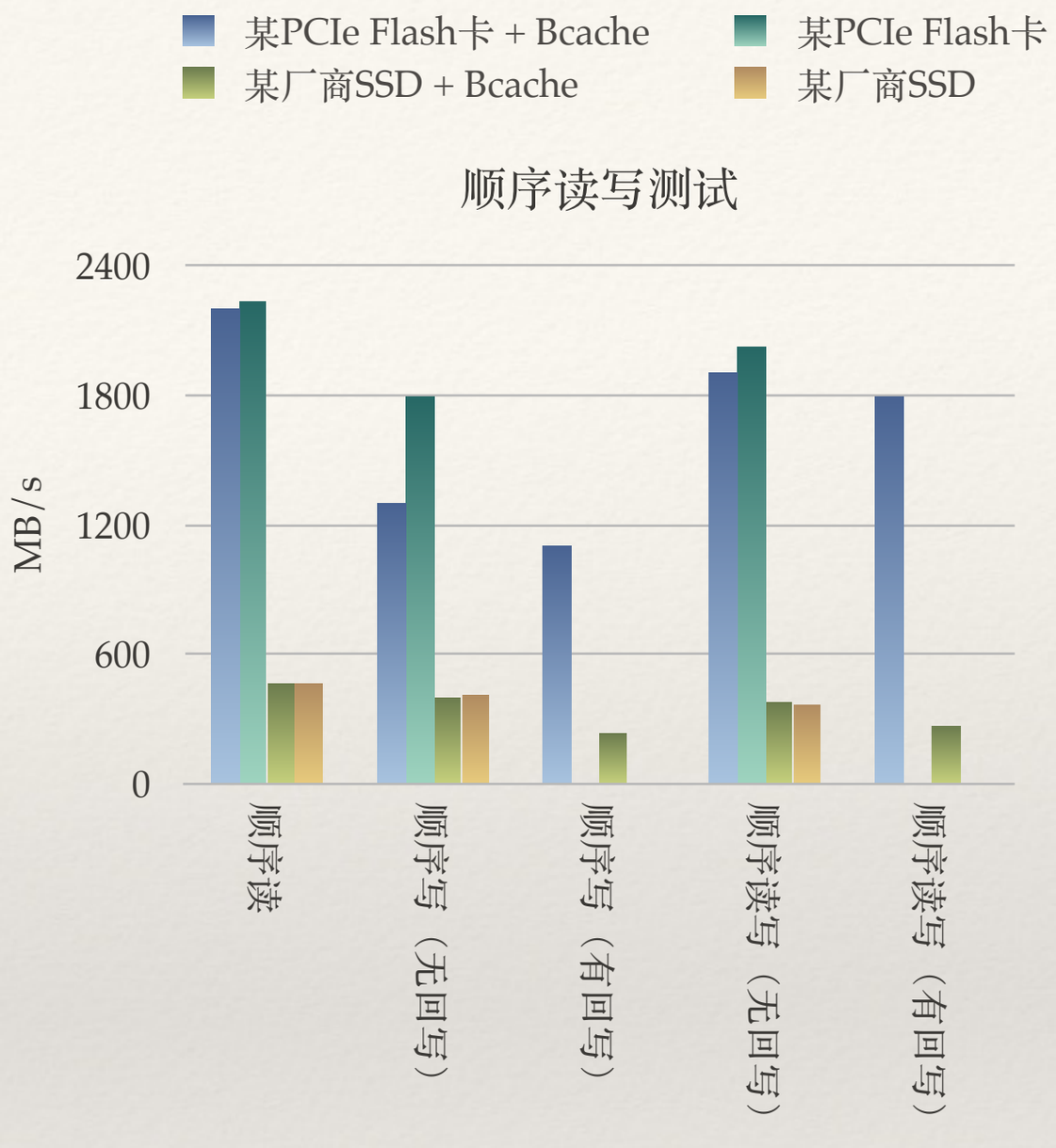
- 写回还是写透?
- 写回式缓存可带来很高性能，然而在主机掉电或崩溃时数据虽然还在SSD上持久化，但仍然只能被视为丢失
  - 阿里云ECS的“宕机迁移”
- 限制每个虚拟机在ssd上的脏数据大小，超过后强制写透
- 更激进的回写策略
  - 对整体性能有一定影响
- BIO_FLUSH/BIO_FUA时等待脏数据清空，维持posix的fsync语义
- 用户是否能承受宕机掉电时的数据损失?
  - 用户自己决定

产品化(4)

- 某些SSD在IO不按4KB对齐时性能明显下降
  - 这里说的“对齐”，是指同时在偏移和大小上对齐
- Bcache会尽量bypass不按physical block size对齐的写，然而……
  - 不对齐写如果和正在回写的数据有交集，就必须写到SSD上
  - Windows有大量不对齐写
- 大小不按4KB对齐的数据块写在SSD上之后会随机影响所有后续写请求
  - 原因：所有数据块在一个512KB桶内连续放置

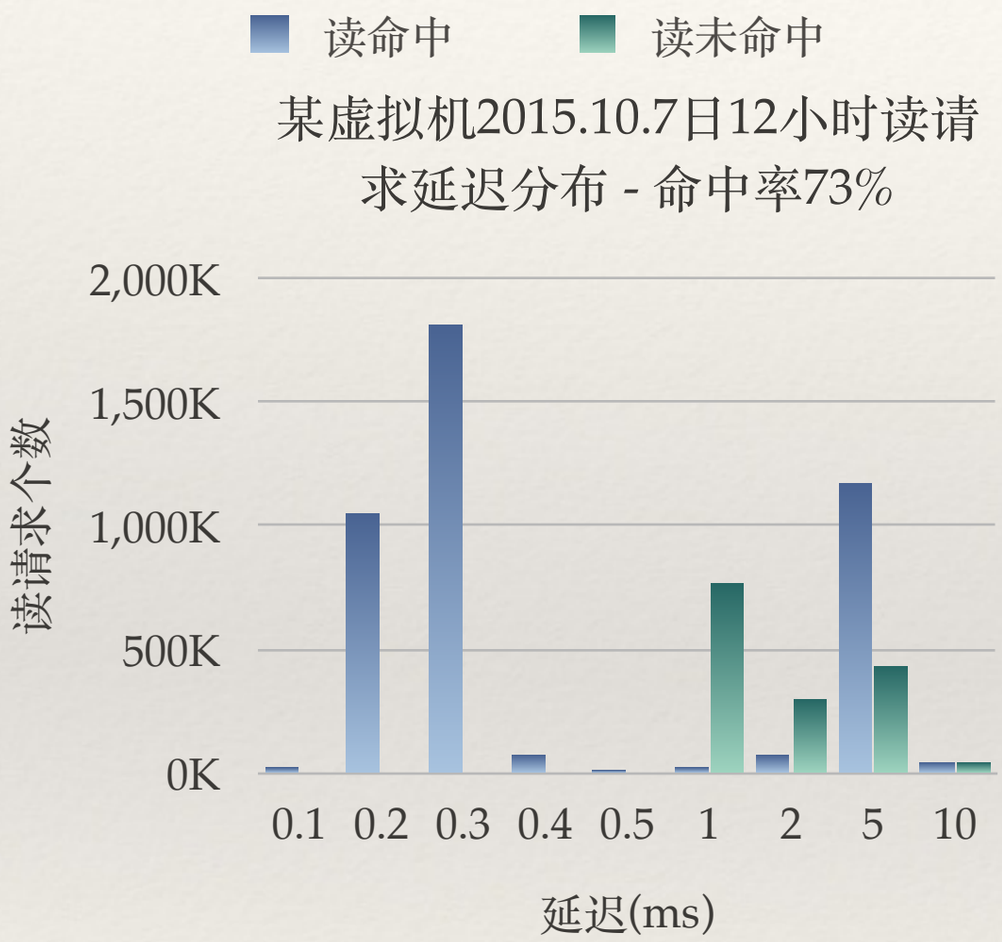


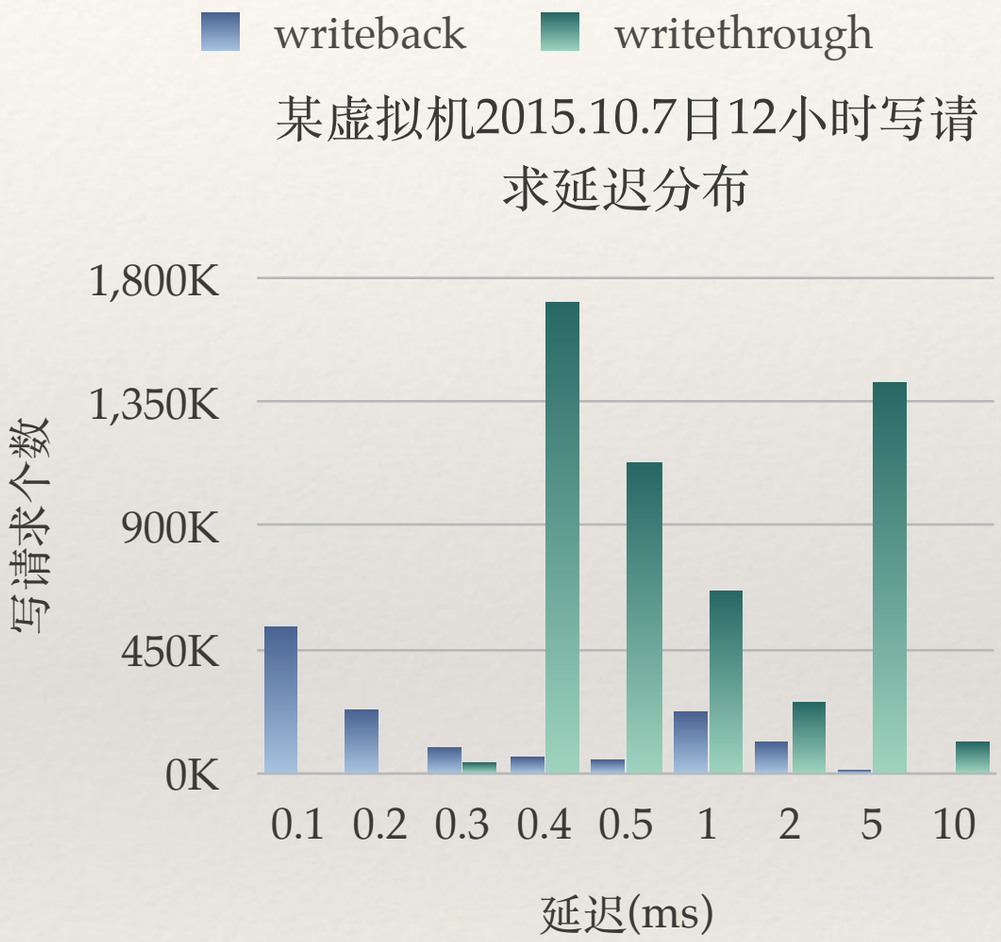




* 顺序读写测试: bs=512k, iodepth=8, numjobs=2
* 随机读写测试: bs=4k, iodepth=32, numjobs=2
* 以上测试的后端设备均为一块1.2TB sata硬盘
* 裸设备的对比不适用于有回写的情况

延迟分布


读命中
读未命中
某虚拟机2015.10.7日12小时读请求延迟分布 - 命中率73%
2,000K
1,500K
1,000K
500K
0K
读请求个数
0.1
0.2
0.3
0.4
0.5
1
2
5
10
延迟(ms)


writeback
writethrough
某虚拟机2015.10.7日12小时写请求延迟分布
1,800K
1,350K
900K
450K
0K
写请求个数
0.1
0.2
0.3
0.4
0.5
1
2
5
10
延迟(ms)

What's Next?

- 精确定义并维持服务质量
- 如何利用下一代高速设备
  - Persistent Memory使大多数软件成为瓶颈
  - 块设备界面
  - 存储虚拟化逻辑以及数据平面下沉到硬件上?
- 更好的淘汰策略
  - LIRS
- 万兆网或者更高速网络连接下的分布式存储是否还需要本地缓存?