谈PCIe ssd在数据库优化中的作用II之颠覆性创新

Apr 2015

Shannon Systems

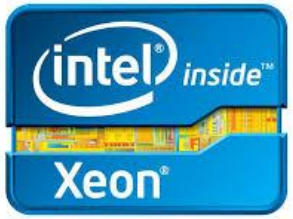
宝存科技

Shannon Systems

宝 存 科 技

关于摩尔定律

- 由于基于Flash闪存存储设备的出现;处理器,内存和存储设备终于都可以遵循摩尔定律实现快速的发展.



12 核心, 24 线程
@3.2Ghz

DDR4 DRAM@2133MHz
768GB @2U

500K IOPS, 9us 延迟
50TB 裸容量 @2U

当谈及数据库应用的存储方案时

- 容量
- 性能
- 安全性
- 应用成本
- Flash 存储起到的作用

基于Host的PCIe Flash存储的优化与方案

- 超大容量高可用Flash存储(Ultra-capacity and highly available Flash)
- 原子写(Atomic Write)
- Redo log 优化(Redo log optimization)

大容量Flash存储(Ultra Capacity Flash Storage)

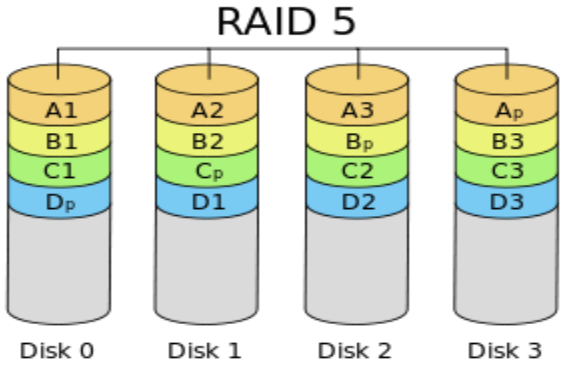
- 常规方案:
  - HBA
  - 软RAID

- 缺点
  - 不支持Flash存储设备的高级特性, 比如Trim, S.M.A.R.T信息等.
  - 性能损耗
  - 容错性差,风险高, 如意外掉电数据安全性问题,连续故障等.

大容量Flash存储(Huge Capacity Flash Storage)

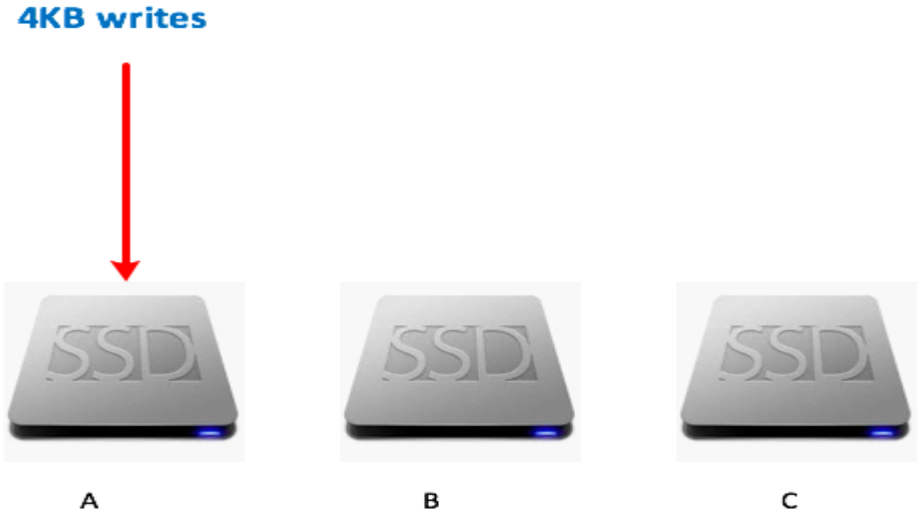
- 轮转式校验数据存储，最多可有一个SSD故障

- 由于没有校验数据缓存,数据的随机写入会导致:
  - 在系统层面,写放大系数一定大于2(过大).
  - 更快速的寿命损耗.
  - 会出现”写洞”现象，影响数据一致性.



RAID5处理随机数据写入的步骤

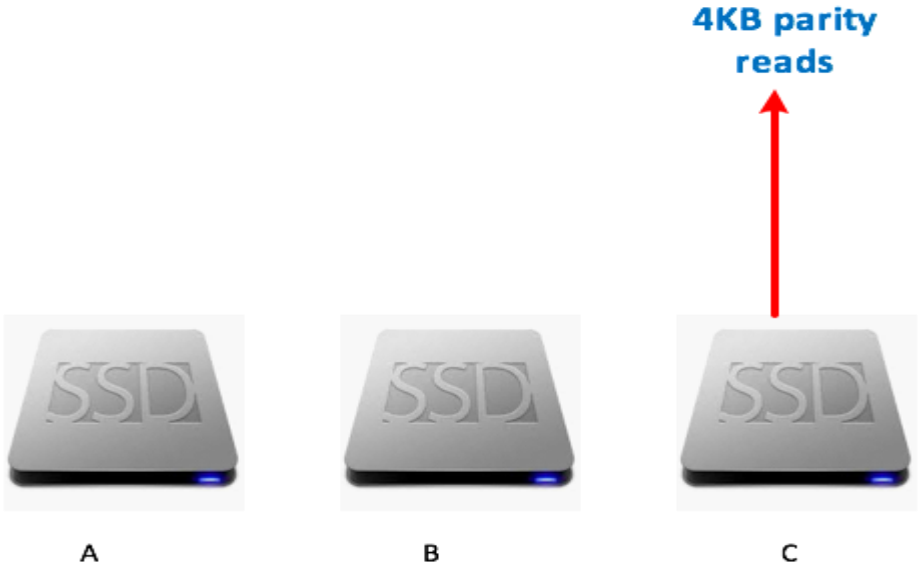
- 4K 随机写第一步：写入待写数据



当磁盘A被数据填满时,会发生写放大现象

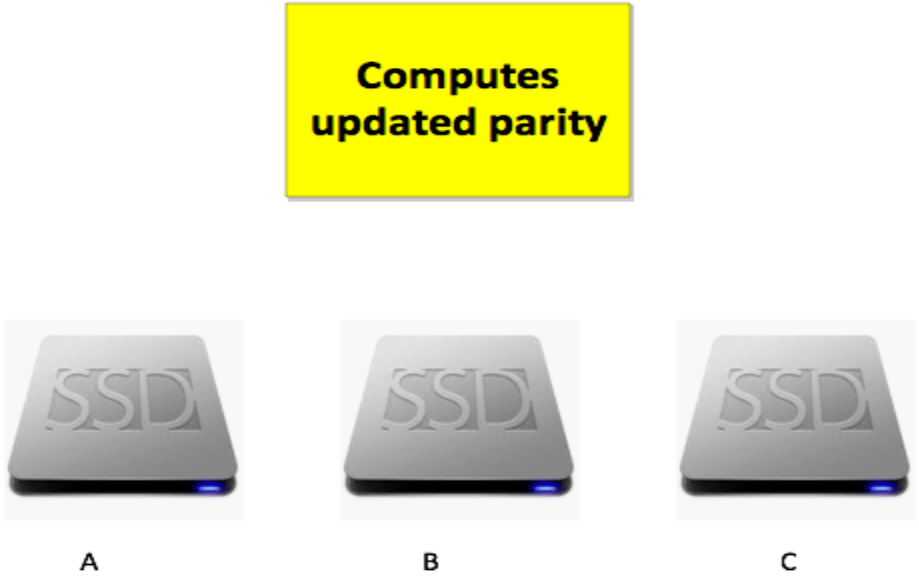
RAID5处理随机数据写入的步骤

- 4K 随机写第二步:读取原校验数据



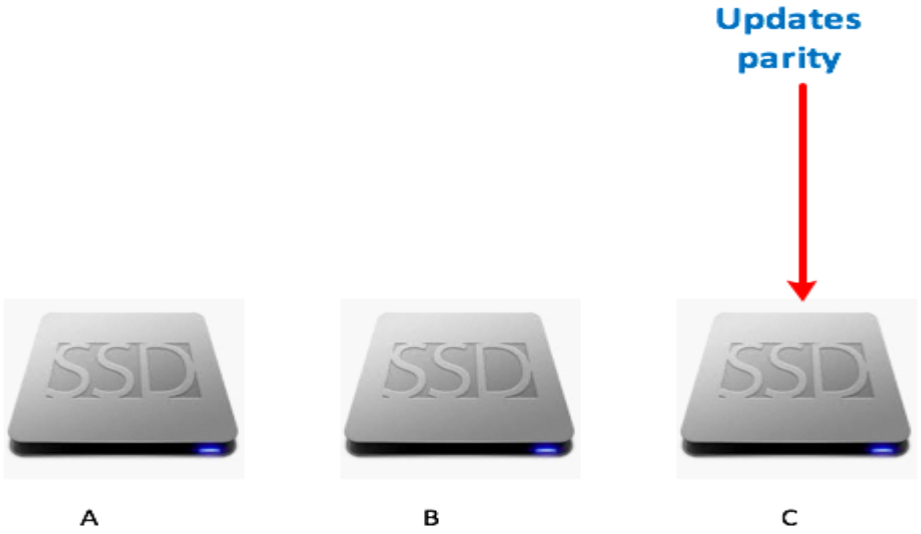
RAID5处理随机数据写入的步骤

- 4K 随机写第三步:计算出新的校验数据



Random writes in RAID-5

- 4K 随机写第四步:更新校验数据

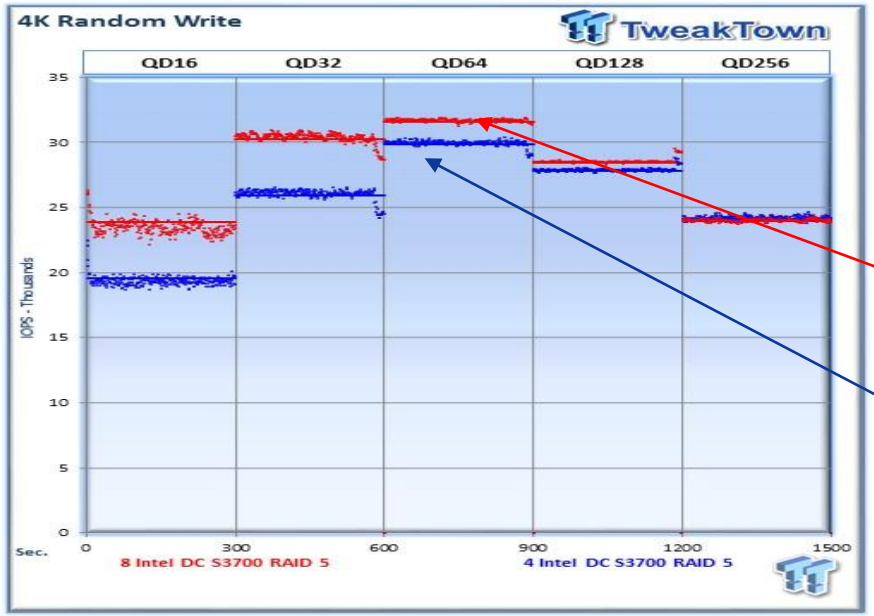


大容量Flash存储(Huge Capacity Flash Storage)

- 由于 RMW(read-modify-write), RCW(read-reconstruct-write) 导致了物理双写, 使整个Array 的WAF(写放大因子) 远远大于2
- 在 RMW, RCW 过程中如果系统出现其他故障可导致交验数据更新不成功, Write Hole(写洞) 现象产生, 带来数据一致性问题.

大容量Flash存储(Huge Capacity Flash Storage)

- 随机写性能不是随着阵列中的磁盘数量线性增长.



8 盘 (RAID-5)

VS

4 盘 (RAID-5)

PCIe-RAID

PCIe-RAID

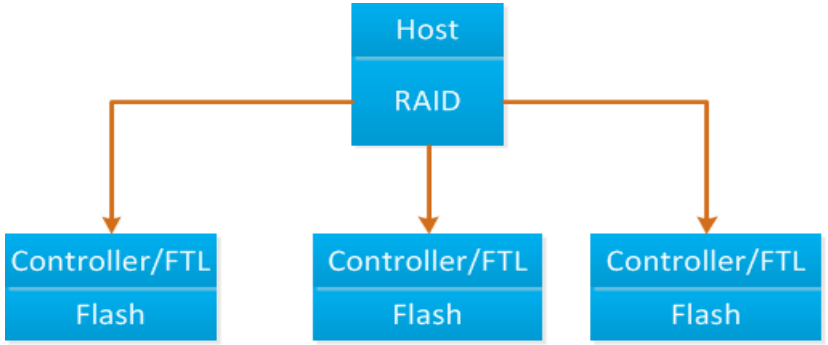
- 目的:
  - 满足应用程序的大容量需求
  - 避免单点故障(硬件故障,多发性Flash芯片失效,主控失效等.)
  - 解决传统的RAID5阵列写性能极差的问题
  - 解决传统的闪存盘阵列会有很大的写放大现象.
  - 解决传统SSD硬盘性能稳定性和性能一致性不足的问题
- **目标: 在系统中提供一个集大容量,高性能和高可靠性的逻辑块设备.**

PCIe-RAID

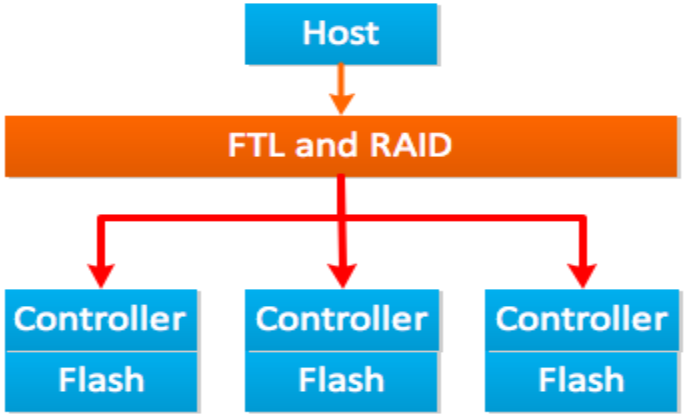
- 技术关键点:
  - 软件FTL层
  - 跨设备FTL层
  - 基于PBA的RAID
  - 采用2维RAID,以达到最大的保护效果

PCIe-RAID

- Host-Based 将FTL的实现从Flash存储设备中移至主机
- 统一FTL和RAID层,可以解决传统RAID存在的诸多问题



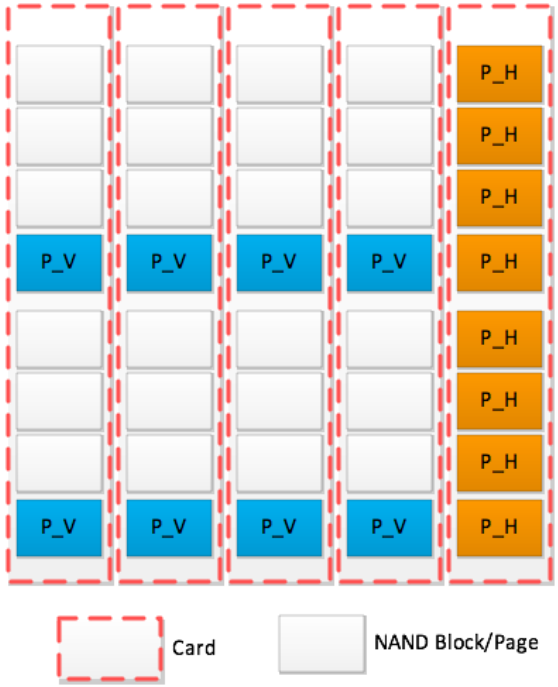
Conventional RAID5 of SSDs



The new RAID5 architecture

PCIe-RAID

- 2维RAID，最大化数据保护



Demo

PCIe-RAID

- 性能(保守):
- 4K RW　　30W+ IOPS
- 4K RR　　50W+ IOPS
- 延迟:　R:80us / W:15us
- 冗余度/维护:
- PCIe RAID 5 允许一个PCIe Flash设备彻底失效.
- 未来有可能支持RAID10
- PCIe 接口支持热备设备, 8639接口支持热插拔,热维护.
- 综合OP可以最多释放到15%以下,更大的用户空间

PCIe-RAID

- 高密度
  - 2U 服务器，最多可以部署6张全高的PCI-E 板卡，如 HP DL380 Gen9
  - ~50TB 裸容量，最高40TB的用户可用容量
  - 3U 服务器，最多可以部署11张全高的PCI-E 板卡，如 Supermicro Gen X9DRX+-F
  - ~90TB 裸容量，最高80TB的用户可用容量

PCIe-RAID

- 优势:
  - 大容量,高性能
  - 全局垃圾回收GC和磨损均衡(WL)
  - 基于PBA的RAID 构建实现, RAID5的全局写放大系数远远小于2, 更高的Flash 寿命
  - 避免RMW/RCW
  - 避免传统RAID5所带来的性能损耗.
  - Host-Base 的UniFTL 可以感知校验数据状态,避免”写洞”(WriteHole)

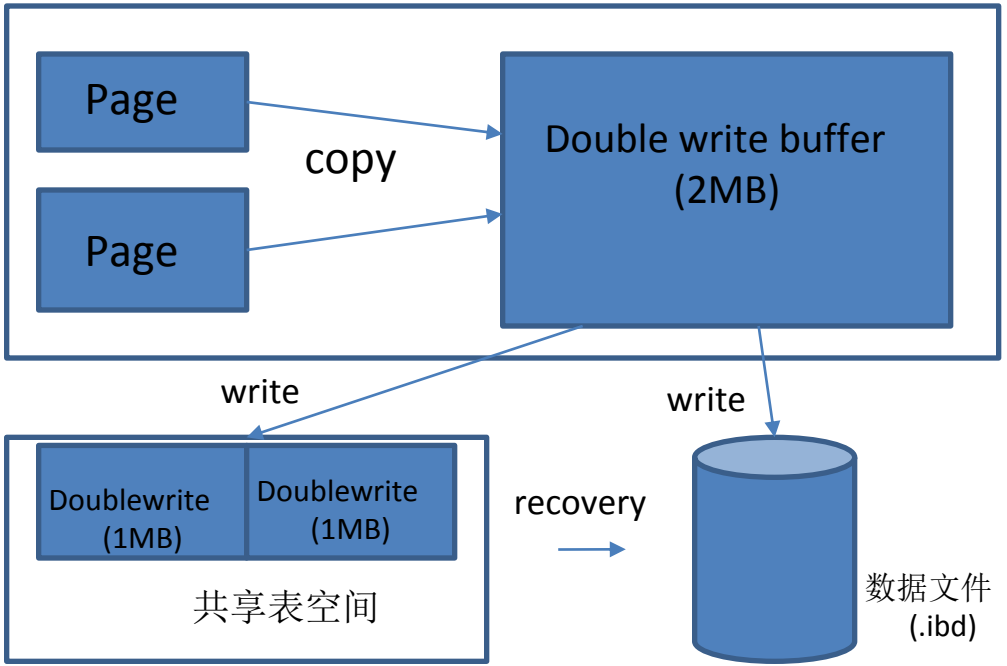
原子写(Atomic Write)

- 原子操作：读/写
- Page 和 Buffer.
- Flash Page：Flash 的原子读取，写入单位，多以4K为例，但是在现代Flash产品中 实际多为16KB/32KB.
- InnoDB Page：是InnoDB 的原子读取,写入单位. 一般为16KB.
- PCIe Flash Write Buffer(以Direct-IO产品为例)：主控中的SRAM，3组,每组32KB.
- InnoDB Double Write buffer：主机内存中的空间,用来存储Double Write 数据，2MB

原子写(Atomic Write)

- 传统硬件和操作系统不能保证InnoDB Page 写入这一操作的原子性
- InnoDB使用Double write这个特性来避免InnoDB Page写入不完整的问题.

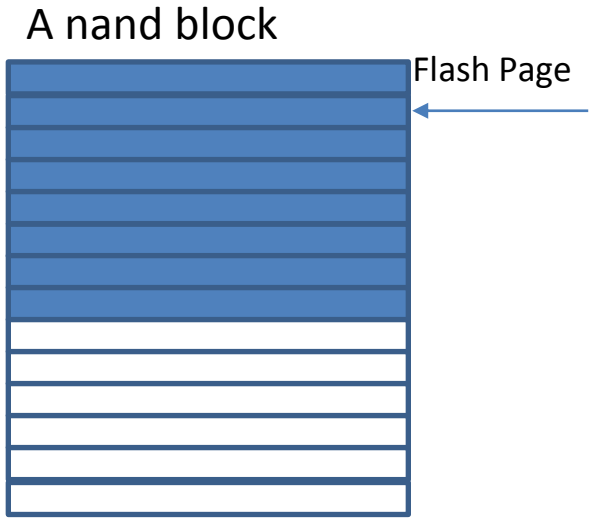
- Double Write 缺点:
- 数据写入量加倍(Bad For Flash)
- 增加了写入负载(不是2倍)



原子写(Atomic Write)

- InnoDB Page Size: 16KB
- Flash Page Size: 32KB
- 对NAND Flash 闪存的页进的行读写操作都是原子操作.

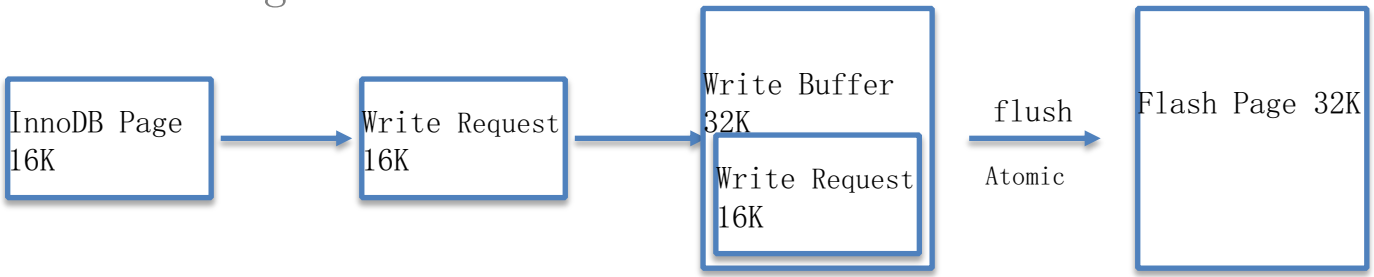
- 如果我们确保将每个InnoDB Page 一次性的写入一个Flash Page(InnoDB Page Size<= Flash Page Size),那么InnoDB Page 写入就是原子操作.
- 保证InnoDB Page 数据存储不跨 Flash Page.



原子写(Atomic Write)

- Direct-IO 产品中原子写的实现

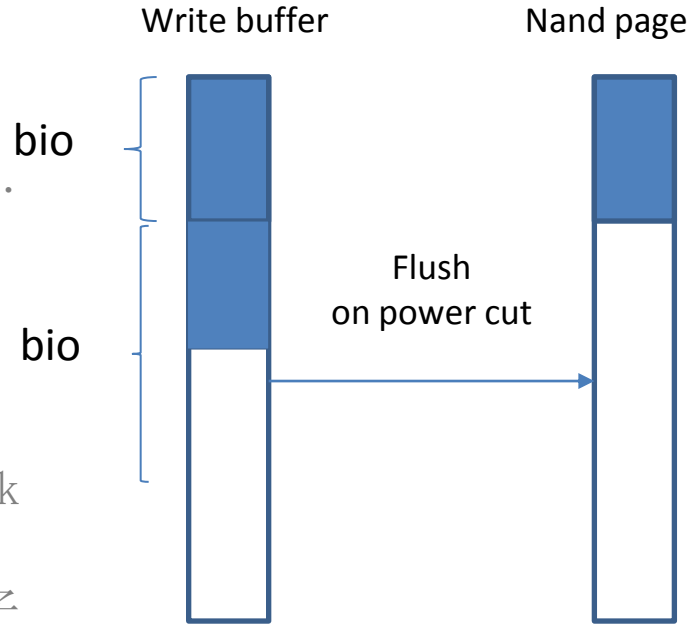
- InnoDB Page Size 16KB
- InnoDB Write Request Size 16KB (多数情况)
- Flash(主控) Write Buffer Size 32KB *3
- Flash Page Size 32KB



原子写(Atomic Write)

- 当Request Size <= Flash Write Buffer(32KB)时
- Flash Write Buffer 为空时
- 所有数据先入Write Buffer, 再刷入Flash Page.

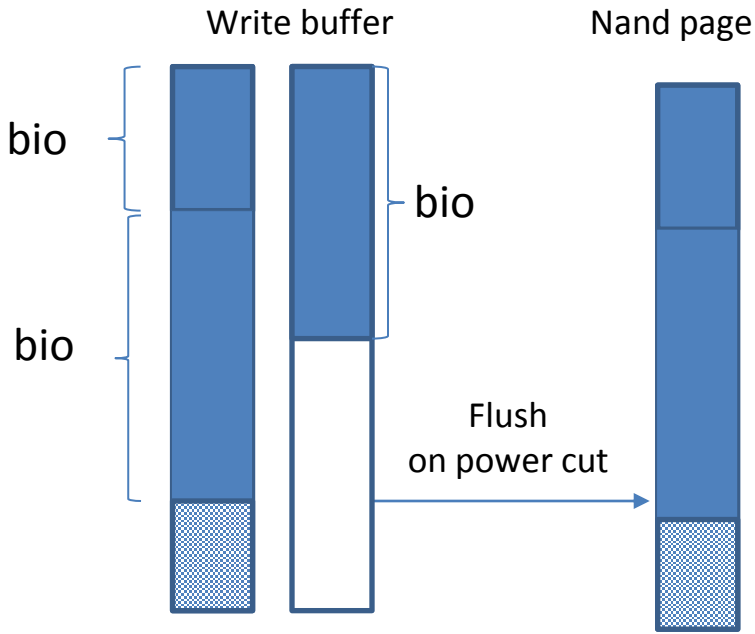
- 在突然断电时,只有完全写入了Flash Write Buffer的BIO(block io) Request会被刷入NAND闪存的页
- 没有完全被写入Flash Write Buffer的bio(block io)会被丢弃.
- 保证小于32KB(InnoDB Page 16K)写入操作的原子性.



原子写(Atomic Write)

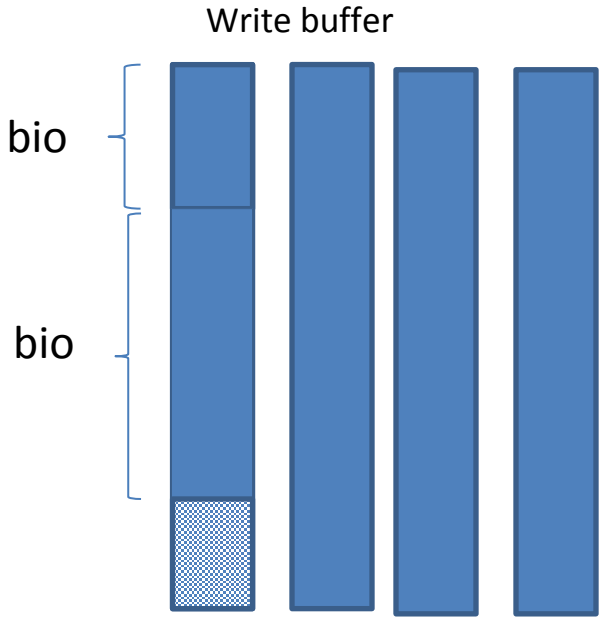
- 如果 Flash Write Buffer 是Half Full的.
- Write Request Size <= Free Space of Flash Write Buffer
- 同上

- Write Request Size > Free Space of Flash Write Buffer
- 将本组Write Buffer 剩余空间用垃圾数据填充，新开一组Write Buffer 来承接这个新的Write Request.



原子写(Atomic Write)

- 有时连续的写请求会被合并为一个大BIO(Block IO)
- Write Request > Flash Page Size 32K
- 数据会不经过Write Buffer被直接写入一个新的 Flash Page(延迟稍高).



原子写(Atomic Write)

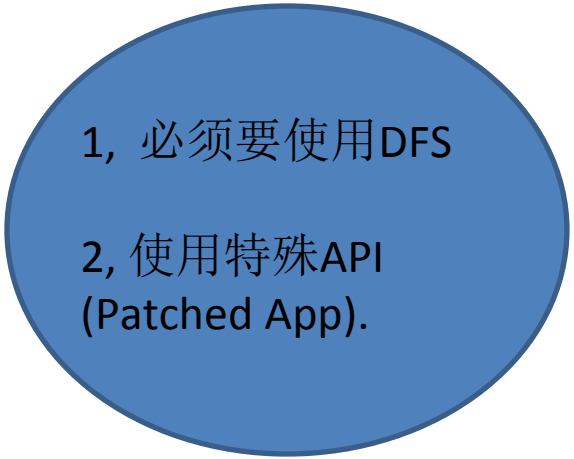
- 应用原子写的条件
- Write Request Size <= Write Buffer Size 32KB
- 16KB <= 32KB √
- 不要在MySQL/InnoDB/文件系统/块设备层面,分割/合并任何BIO
- O_DIRECT, Double Write=0 √
- Flash FTL 可感知 BIO 的相关信息, Host-Base Flash Only.
- Shannon 或 Fusion-IO(SanDisk) √

原子写(Atomic Write)

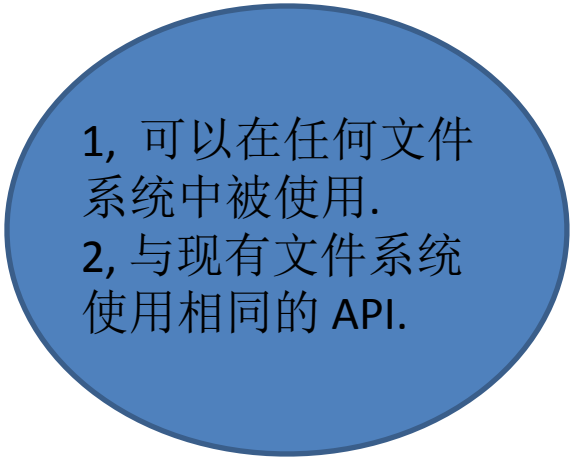
- 应用原子写的效益
  - Double Write = 0
  - 性能提升TPS ~10% (Shannon Systems Lab)
  - 延迟降低 ~50% (Shannon Systems Lab)
  - Flash存储产品的寿命 200%, 可靠性增强.

Shannon vs Fusion-IO

Fusion-IO



Shannon



Redo Log 优化(Redo log optimization)

- InnoDB 具有先写日志特性. redo log 落地后, 就认为数据库完整.
- redo log: 写入单位512 Byte, 顺序写.

- Flash Storage(以Direct-IO 产品为例)
- Write Buffer: 32KB *3
- Direct-IO 主控: 2 Data Pipelines
- 每个 Data Pipeline 独占一组 Write Buffer 32KB
- 2个 Pipelines 共享最后一组 Write Buffer, 抢占, 锁
- 数据入Write Buffer ,向上汇报写入完成, 后台Flush Data to Nand, 掉电保护由硬件完成.

Redo Log 优化(Redo log optimization)

- 我们能做什么?
- 如果我们预留一组Pipeline 和 Write Buffer. 专门处理redo log
- 在Driver 中调高 此种BIO 的优先级.
- 期望收益: 性能提升

- 问题:
- 如何将redo log 的IO请求, 与其他的IO请求进行区分?
- 如何让Driver 感知特殊类型的IO?

Redo Log 优化(Redo log optimization)

- 解决方法:
- 在应用/FS层对redo log 的IO请求加入特殊的Flag, 用来识别这种特殊IO请求.
- Patch MySQL/InnoDB
- Patch FS/EXT4

Redo Log 优化(Redo log optimization)

- Flag 的选择
- Linux 系统中所有的io请求都会用 summit_bio()来进行下发
- bio(bio_request)是在<linux/bio.h>中定义的结构体.

Redo Log 优化(Redo log optimization)

- bio 数据结构

```
struct bio {
        sector_t            bi_sector;        /* associated sector on disk */
        struct bio          *bi_next;         /* list of requests */
        struct block_device *bi_bdev;         /* associated block device */
        unsigned long       bi_flags;         /* status and command flags */
        unsigned long       bi_rw;            /* read or write? */
        unsigned short      bi_vcnt;          /* number of bio_vecs off */
        unsigned short      bi_idx;           /* current index in bi_io_vec */
        unsigned short      bi_phys_segments; /* number of segments after coalescing */
        unsigned short      bi_hw_segments;   /* number of segments after remapping */
        unsigned int        bi_size;          /* I/O count */
        unsigned int        bi_hw_front_size; /* size of the first mergeable segment */
        unsigned int        bi_hw_back_size;  /* size of the last mergeable segment */
        unsigned int        bi_max_vecs;      /* maximum bio_vecs possible */
        struct bio_vec      *bi_io_vec;       /* bio_vec list */
        bio_end_io_t        *bi_end_io;       /* I/O completion method */
        atomic_t            bi_cnt;           /* usage counter */
        void                *bi_private;      /* owner-private method */
        bio_destructor_t    *bi_destructor;   /* destructor method */
};
```

Redo Log 优化(Redo log optimization)

- unsigned long bi_flags; /* status and command flags */
- 无符号长整型，64bit. 约定第16bit 设置为1 来标记redo log 的io 请求
- EXT4_PRIO_FL=16, MySQL/InnoDB Flag
- BIO_RW_PRIO=16, FS/EXT4 Flag

Redo Log 优化(Redo log optimization)

- Patch MySQL/InnoDB
- os_file_create_func
- os_file_set_nocache 设置redo log 文件为O_DRIECT
- 判断文件类型，如果为OS_LOG_FILE 设置Flag，使用ioctl将Flag交给FS
- flags = flags | EXT4_PRIO_FL;
- ioctl(file, EXT2_IOC_SETFLAGS, &flags);

Redo Log 优化(Redo log optimization)

- Patch FS/EXT4
- 判断MySQL/InnoDB 传下来的flag
- if (EXT4_I(inode)->i_flags & EXT4_PRIO_FL)
- 设置Flag 传给Direct-IO Driver
- bio->bi_flags = bio->bi_flags | (1 << BIO_RW_PRIO);

Redo Log 优化(Redo log optimization)

- Flash Storage Driver
- 保留一个 Data Pipeline 和 一组Write Buffer(二者绑定)
- 判断 bio->bi_flags & (1 << bio_rw_prio)
- 将符合条件的BIO 优先下发给保留的Pipleline 和 Write Buffer.

Redo Log 优化(Redo log optimization)

- redo log 优化应用条件
- Patched MySQL/InnoDB √ 改动不会超过20行.
- Patched FS/EXT4 etc. √ 改动不会超过20行.
- Shannon Direct-IO PCIe SSD √
- Enable Atomic Write Feature when Direct-IO Driver loads
- Enable redo log optimize Feature when Direct-IO Driver loads

Redo Log 优化(Redo log optimization)

- redo log 优化效益
- 性能提升 > 100% (In Shannon Systems Lab)

- NOTE:
- 如果没有Patch MySQL/InnoDB FS/EXT4 就在Direct-IO Dirver中开启redo log optimize 特性
- Direct-IO 产品性能表现会变差, 因为资源一直被预留不被使用.
- 开启redo log optimize 特性后, 如果直接关闭该特性.
- 会有非常小的几率引发一个Flash Page 的数据一致性问题.
- Shannon Systems 建议在关闭之后reformat Flash Storage

当谈及数据库应用的存储方案时

- 容量
- 性能
- 安全性
- 应用成本

当谈及数据库应用的存储方案时

- 容量:
- 6.4TB/12.8TB 单设备（大容量）
- PCIe RAID(超大容量)

当谈及数据库应用的存储方案时

- 性能:
  - 原生PCIe Flash 本身具有的超高性能.
  - 原子写(10%)
  - redo log 优化(>100%)

当谈及数据库应用的存储方案时

- 安全性:
- 原子写(写入完整性)
- PCIe RAID(单点,写洞)
- 掉电数据保护

当谈及数据库应用的存储方案时

- 应用成本:
- Flash Storage 成本优化空间
- 高密度,大整合比
- 性能提升带来性价比提升

当谈及数据库应用的存储方案时

- 容量 √
- 性能 √
- 安全性 √
- 应用成本 √

Q&A

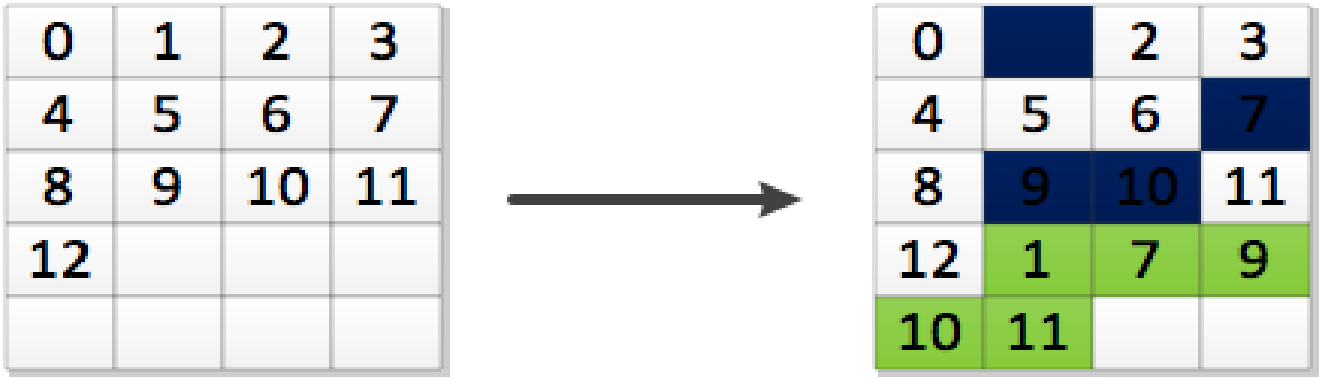
上海宝存信息科技有限公司
Shannon Systems

上海市杨浦区大连路588号宝地广场A座305室
Suite 305 Tower A Baoland Plaza 588 Dalian Road Yangpu Shanghai 200082
021-5558-0181
contact@shannon-sys.com
www.shannon-sys.com

Thanks for your time.

Flash存储中的LBA(逻辑地址)与PBA(物理地址)动态映射

- 传统的RAID控制器会仍然将Flash存储视为普通的传统硬盘.
- 存储与传统硬盘有着本质的区别,即将LBA动态映射给PBA.

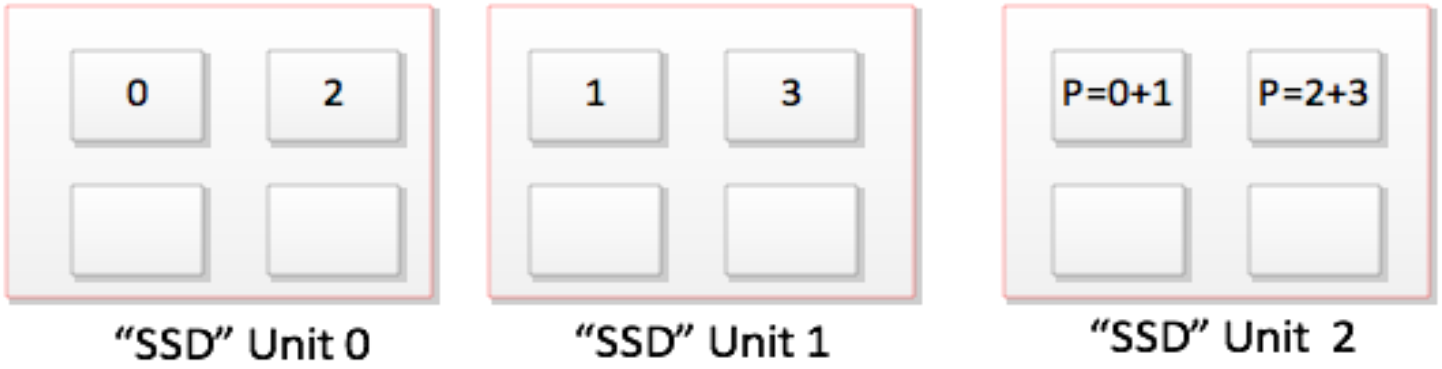


存储中的非动态因素

- 在Flash存储中,物理块地址(PBA)是”静态”因素
- 可以基于”静态”的PBA来构建RAID
- 将FTL的实现从Flash存储中移至主机中将特别有利于基于PBA的RAID阵列的构建.

在FTL层构建RAID的原理（1）

- 初始状态
  - 数字代表LBA
  - 区块代表PBA



在FTL层构建RAID的原理(2)

- 后续状态
  - 数字代表LBA
  - 区块代表PBA