|  |
| --- |
| 总结 |
| Primary storage deduplication需要考虑两点   * Write/read latency * 去重率   其中write/read latency为首要考虑，因此，需要非常注意减少write path和read path上的开销  方案：   * Inline deduplication * Offline deduplication   + Space efficiency 降低   + 后端去重会与前端的application操作产生一定的冲突，降低前端application的执行效率   现有方案总结：   * 现有的方案为了减少read performance的损失，总是选择针对sequence而不是single chunk进行去重   文件可以根据自身可被去重的性质分为三类：   * highly-deduplicatable-type：例如document, virtual machines * poorly-deduplicatable-type: audio, video, compressed files, encoded and encrypted files * unpredictabletype (U-Type): other file types   思考：对audio和video进行chunk level方面的去重是不是很划不来？ |

|  |
| --- |
| [iDedup: latency-aware, inline data deduplication for primary storage.](https://static.usenix.org/events/fast/tech/full_papers/Srinivasan2-10-12.pdf) |
| 场景:   * Primary storage. 这类场景**对延迟很敏感**(用在备份系统则是对throughput以及restore speed比较敏感) * 对primary storage去重时需要面对的挑战   + 去重天然就造成碎片化，导致读的时候需要多次查询，导致读延迟增加   + 写的时候需要计算hash，需要和已存的hashes进行比较，若为重复块时需要增加reference count，并且当数据量较大时需要读取disk上存的hashes，这会造成写延迟增加   该场景下数据流特点：   * 读：temporal locality exists in the access patterns of duplicated data.   + 意味着可以在内存中存有cache来减少latency * 写：spatial locality exists in duplicated primary data   方案：   * NVRAM用以吸收用户新的写，从而减少latency。满足一定条件时，将数据去重后写入disk，这一过程又称为destage * 对文件进行分块操作，去重是为了eliminate duplicate sequences，而不是针对chunk   + 这是bulk operation，减少了写延迟   + 这牺牲了去重率，但是减少了数据的fragmentation，利于减少读延迟 * 根据temporal locality，内存中有LRU cache，记录了数据块与其存储位置，用以去重 * 用于去重的iDedup algorithm(采用的数据结构还未怎么理解)：   + Sequence identification: Identify duplicate block sequences for file blocks.   + Sequence pruning: Process duplicate sequences based on their length.   + Sequence deduplication: Deduplicate sequences **greater than the configured threshold**   系统示意图： |
| Datasets:  A. W. Leung, S. Pasupathy, G. Goodson, and E. L. Miller. Measurement and analysis of large-scale network file system workloads. In Proceedings of the 2008 USENIX Annual Technical Conference, pages 213–226, June 2008. |
| 优点：  缺点：   * Sequence length是人为指定的，需要人工设定，并不一定适合所有数据集 |

|  |
| --- |
| POD: Performance Oriented I/O Deduplication for Primary Storage Systems in the Cloud |
| 背景：   * 直接对primary storage采用deduplication需要占用一定的内存空间，造成数据碎片化。前者时因为需要缓存chunks的fingerpirnts等数据，占用一部分内存；后者碎片化是去重一定会导致的，并且会使得原先可以顺序读的数据出现随机读的特征。 * Primary storage workload的特点是small I/O request很多。如果可以通过去重系统去掉这些small I/O requests，则整个系统的性能得以提升 * Primary storage workload具有burstness特性，即read/write requests突然升高。此时read cache和index cache形成对内存空间的竞争   Select-Dedupe：   * Chunk sequence一共分为三类：1. 全部为重复块 2. 部分为重复块，但是重复块数量小于一个阈值 3. 部分为重复块，但是重复块大于一个阈值 * 作用为去重，但仅对那些全部为重复chunks或者重复chunks占大部分的write requests进行去重。   + 这是因为针对第二类去重降低的存储空间有限，大部分数据仍要写入硬盘，即marginal effect很小。   + 而对第三类去重则是可以取得很好的节省空间效果，并且写入的数量也有限。 * 组成:   + Index\_table: it maintains fingerprints of the hot data chunks already stored on disks     - LRU cache   + Map\_table: it keeps all the information of the deduplicated write requests whose write data are already stored on disks     - 为了防止丢失，存在Non-volatile memory中 * 流程：   + 当write request出现后，系统将其分成chunks并计算hash，与index\_table中的数据比较，并且分为如上所说的三类。   + 如果为1，3类，系统在map\_table中更新重复chunks的信息，将非重复chunks写入硬盘。如果为第二类，系统直接将所有chunks写入硬盘(未提及此情况下如何更新map\_table)   iCache：   * 为解决read, write出现burstness而设。其功能是根据read, write的变化特点调整index cache和read cache的组成比例。其中index cache用于写数据时查找fingerprints，而read cache则是用于读数据时缓存数据。 |
| 优点   * 与iDedup相比，POD针对小数据进行优化，很多small I/O writes由于去重的原因没必要存到硬盘上 * 由于不对第二类chunk sequence进行去重，平衡了去重率和data fragmentation，读性能下降不那么大   缺点：   * 第二类chunk sequence是需要人为设定threshhold，这很看经验 * 感觉Index\_table这一块是采样检测，估计很多重复情况无法检测出来。并且使用LRU cache无法适用temporal locality不好的情况 |

|  |
| --- |
| Primary Data Deduplication – Large Scale Study and System Design |
| 背景：   * 与backup workload相比时的不同：   + 去重率没有那么高，事实上就是无法做到，因为latency更加重要   + 对去重增加的overhead较为敏感，例如defragmentation、metadata processing等。因此去重需要注意write/read latency，即减少write/read path上额外增加的开销   + 去重不能占用过多的系统空间，否则会影响主要系统的运行。同时去重系统需要与其余软件共享内存、CPU、I/O bandwidth等系统资源。   部分发现：   * 采用variable sized chunk比fixed sized chunk的去重效果更好(shift boudary effect) * Chunk的size越小，去重率越高(但是metadata量增加了)Chunk的size越小，压缩率越低。因此可以选择较大的size+压缩技术以取得很好的空间节约率(可以与采用较小的chunk时的空间节约率相当)   系统实现：   * Hashtable是别人用过的cuckoo hashing with compact keys(Chunkstash里面有) * Chunk metadata采用了log-structured organization * 采用了**post-deduplication**的方案对数据进行去重。 * 为了保证不占用过多内存，该篇文章采用了partition的方式，即将dataset根据文件类型、路径等方式分为一个个partition，去重仅针对partition内部去重(牺牲了压缩率)。但是同时也采用了reconciliation的方式，可以选择性的将各个不同的partition合并。 |
| 缺点：   * File reads有问题(see page 7 file reads)，作者用了compaction的方式对container进行回收，这也导致原先的chunk会分到新的container中。而file recipe中记录的是文件的某个chunk可以在哪个container中找到，在container中的位置(offset)。compaction会使得file recipe中记录的信息失效。而作者是通过per-container redirection table实现的跳转。并且为了可靠性，将这个table存在NVRAM中。这是否影响到了write path？ * 没比较read performance，感觉有问题 |

|  |
| --- |
| Hands A heuristically arranged non-backup in-line deduplication system |
| 场景：   * fingerprints无法全部放在内存中，造成disk bottleneck   方案：   * 根据access pattern来prefetch fingerprints。即先identify working set，然后根据working set来prefetch fingerprints。 |

|  |
| --- |
| DIODE: Dynamic Inline-Offlfline DEduplication Providing Effificient Space-saving and Read/Write Performance for Primary Storage Systems |
| 背景：   * 用在Primary storage的deduplication系统，latency非常重要 * primary storage workload is highly dynamic because of the unpredictable access pattern   方案:   * Inline 阶段：仍然针对sequence进行去重，当duplicate sequence的length超过阈值后才进行去重。sequence length的threshold是动态调整的。调整的公式需要考虑读写请求比例，前一个周期的threshold   + 若写比例增大，则减少sequence length，这是为了减少匹配重复sequence时引起的latency increase   + 若读比例增加，增大sequence length，这是为了减少读时遇到的fragmentation * Offline阶段：如果某文件未被Inline阶段去重，则其被写入硬盘，当文件数目超过一定阈值后，触发post-deduplication   + 每个文件都有优先级，post-deduplication首先对优先级高的进行去重 |

|  |
| --- |
| A Differentiated Caching Mechanism to Enable Primary Storage Deduplication in Clouds |
| 方案：   * Inline阶段：   + 内存中存有fingerprint table，其作用是<fingerprint,LBA> cache；此外LBA mapping table存在battery-backed memory中     - cache采用了TLC-LRU(Temporal Locality Counting-LRU)和TLE-LRU(Temporal Locality Estimation-LRU)。     - 可以看出是绕过了操作系统和文件系统的，直接对硬盘进行操作 * Post processing阶段   + 当system处于idle状态的时候才会触发post processing   + 对Old chunks赋有更高的优先级，这是为了减少与read操作出现冲突的概率   + SSD上的组成：on-disk fingerprint table, on-disk LBA mapping table, reference count table.     - SSD上的fingerprint table 用leveldb存储。     - 在post-deduplication的过程中，为了read performance，仅仅针对超过一定长度的重复sequence进行去重 |