CHINA LINUX KERNEL
中国Linux内核开发者大会

vivo

EROFS 压缩文件 Direct I/O 的探索和支持

郭纯海　vivo文件系统工程师
杨晨志　vivo文件系统工程师

目　录

CONTENTS

Part 1

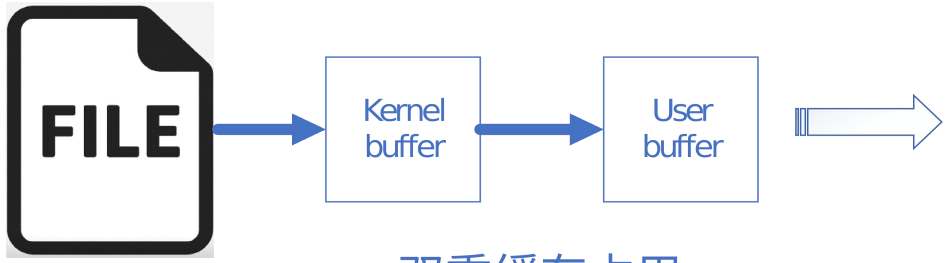
背景介绍

从一个应用问题出发

EROFS（Enhanced Read-Only File System）是安卓只读分区的默认文件系统，相关分区总大小已超过10GB，越来越多的大文件被内置其中。

应用痛点

EROFS 加载只读一次的大文件时，传统 buffer I/O 不仅存在额外拷贝，还引入双重缓存占用，**导致性能抖动和内存回收压力。**理论上 **direct I/O 更契合这种场景**，谷歌甚至针对APEX文件增加过VTS测试强制要求必须支持 direct I/O 以避免此类问题。



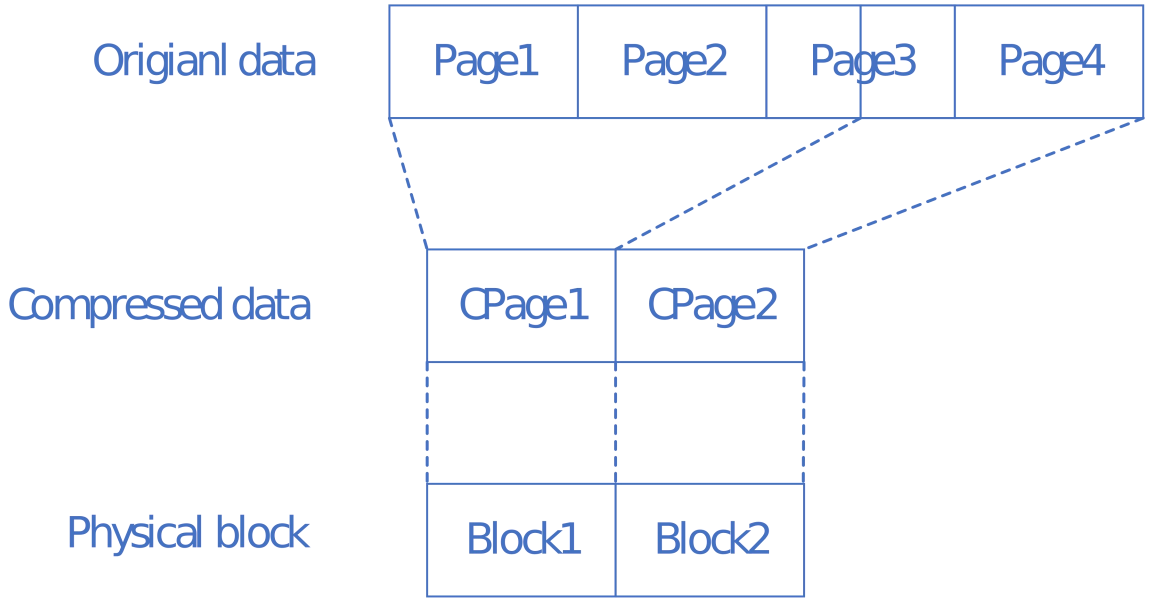
双重缓存占用

```
+// Preinstalled APEX files (.apex) should be okay when opening with O_DIRECT
+TEST(VtsApexTest, OpenPreinstalledApex) {
+  ForEachPreinstalledApex([](auto path) {
+    unique_fd fd(open(path.c_str(), O_RDONLY | O_CLOEXEC | O_DIRECT));
+    ASSERT_NE(fd.get(), -1)
+        << "Can't open an APEX file " << path << ": " << strerror(errno);
+  });
+}
```

EROFS的压缩文件为什么还不支持 direct I/O ？

EROFS 对于压缩文件，目前还不支持 direct I/O，评估主要是考虑到：

- 压缩数据块在磁盘上天然存在非对齐的问题
- 压缩数据在direct I/O随机访问时存在读放大问题，访问效率较低



Part 2

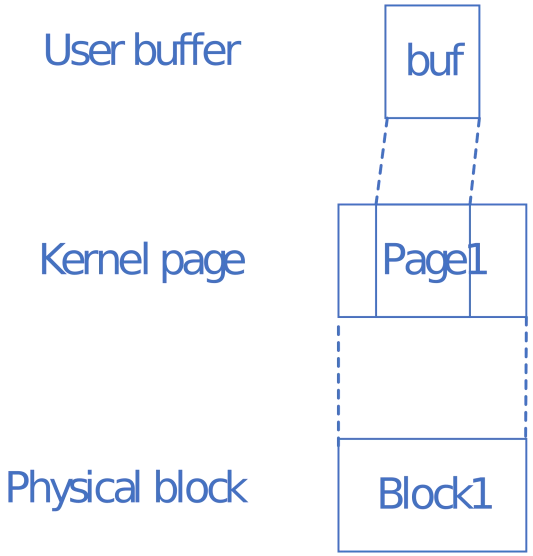
EROFS Direct I/O的实现

传统 direct I/O 为什么要求读取数据和磁盘块对齐？

IO对齐要求源自存储设备：如 UFS存储设备最小传输单位是4KB

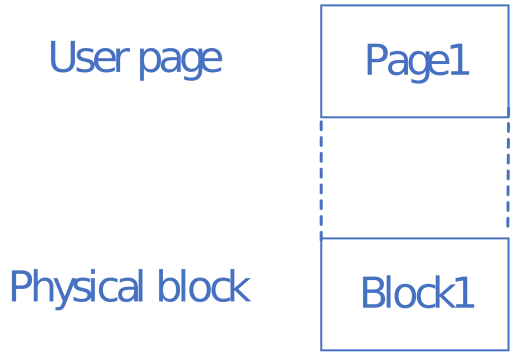
- buffer I/O：

读取user buffer时，因为使用了kernel page作为中转buffer，所以user buffer没有对齐要求



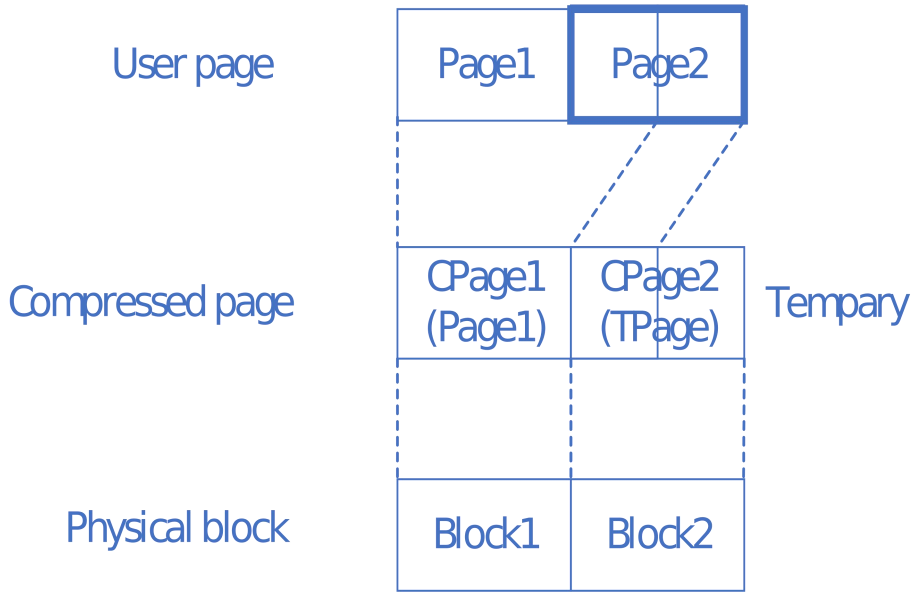
- direct I/O：

读取的user buffer没有kernel page可以中转，所以要求文件偏移、buffer 地址和buffer大小都必须和block大小对齐



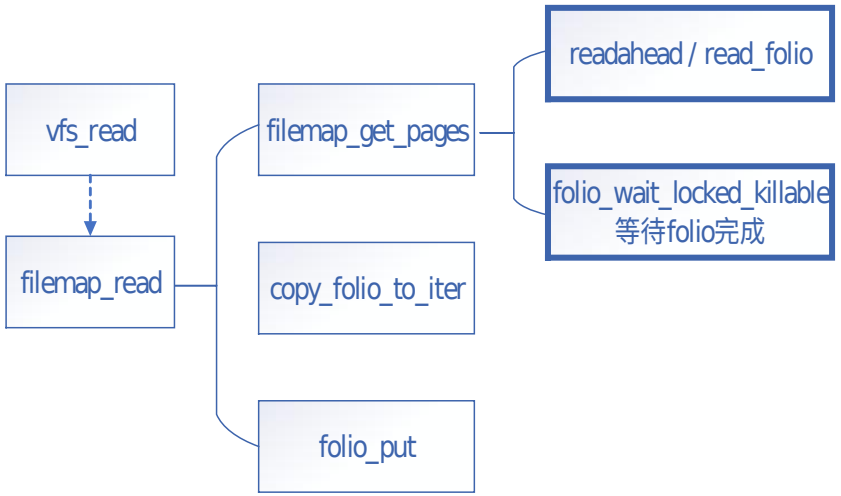
EROFS文件系统怎么支持 Direct I/O ？

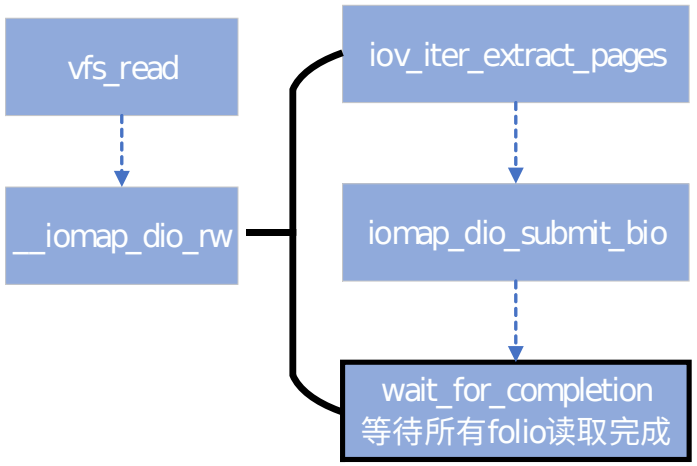
- **EROFS：block地址对齐不是问题**
  - EROFS的实现是先读取到临时page（临时page已经满足了block对齐），再从临时page解压到user buffer中
- **文件偏移和buffer大小：需要block大小对齐**
- **性能思考**
  - 大文件读取是顺序访问，忽略随机访问的读放大影响
  - 顺序访问大部分情况下是就地I/O（inplace I/O）和就地解压（inplace decompression），不需要临时page
- **设计要点：**并发实现，性能调优，原有feature兼容支持等



并发实现（1）—— Buffer I/O和Direct I/O的folio读取对比

- buffer I/O读取时，vfs读取过程中调用 readahead / read_folio 读取folio后，会等待各个folio的完成，所以 readahead / read_folio 对folio的完成处理比较简单（直接unlock folio即可）；
- direct I/O读取相比buffer I/O，如下两方面的并发处理更复杂：
  - 每笔direct I/O请求需要管理各个folio并等其完成，一般会增加结构体进行管理——影响并发
  - direct I/O和buffer I/O并发、多线程direct I/O的并发

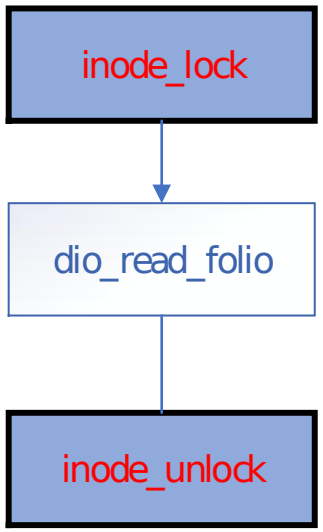


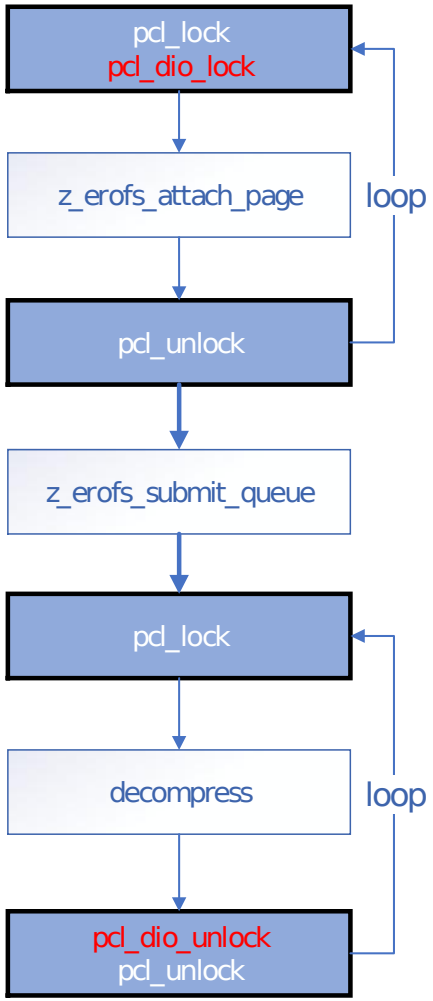


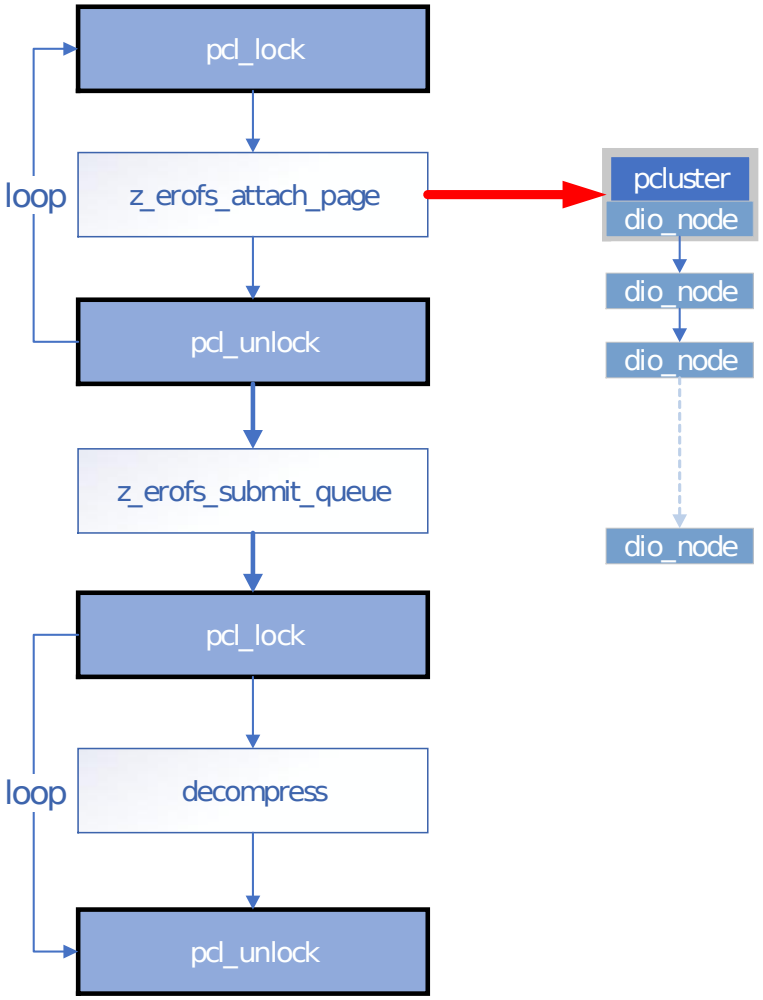

```
struct iomap_dio {
    struct kiocb        *iocb;
    const struct iomap_dio_ops *dops;
    loff_t              i_size;
    loff_t              size;
    atomic_t            ref;
    unsigned            flags;
    int                 error;
    size_t              done_before;
    bool                wait_for_completion;

    union {
        /* used during submission and for
        struct {
            struct iov_iter     *iter;
            struct task_struct  *waiter;
        } submit;

        /* used for aio completion: */
        struct {
            struct work_struct  work;
        } aio;
    };
};
```

并发实现（2）—— EROFS Direct I/O并发设计演进

文件级并发 → pcluster级并发 → 全并发







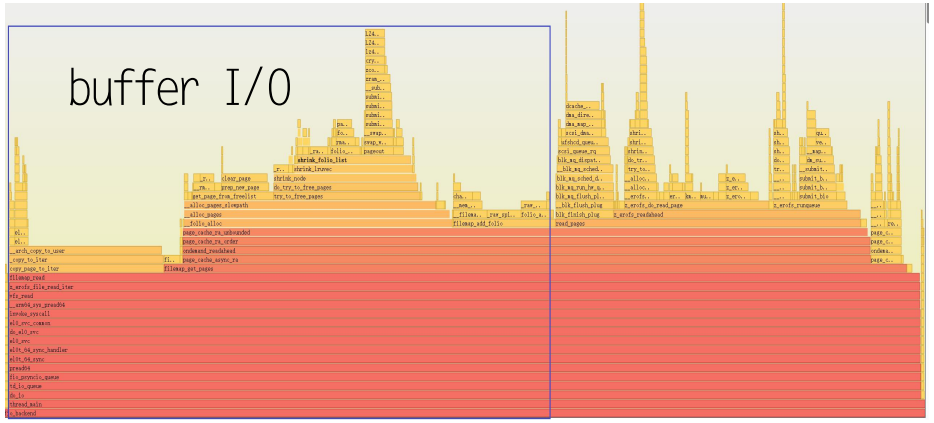
EROFS Direct I/O 8线程FIO测试对比

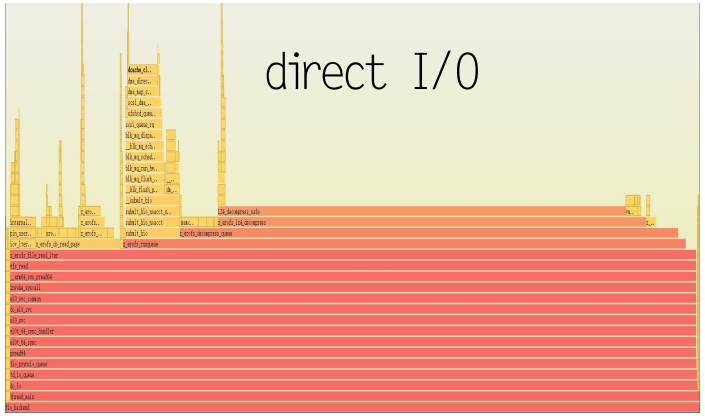
- direct I/O和buffer I/O 8线程的fio测试（读取2.5GB文件）结果如下表，提升了38.7%

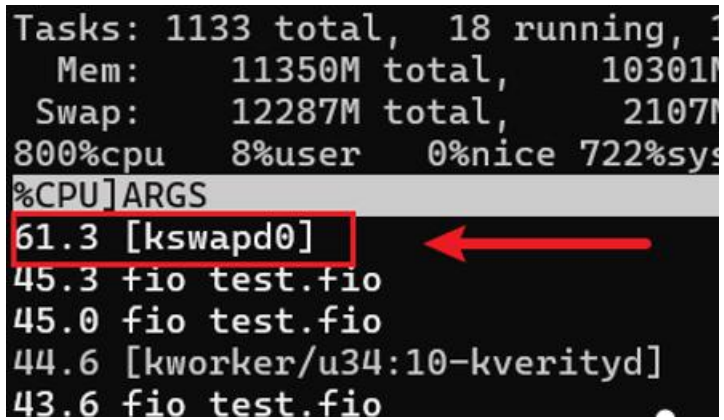
buffer I/O (MiB/s)	direct I/O (MiB/s)	提升比例
2629.8	3648.7	**38.7%**

测试的前期部分是内存充足场景的，低内存场景收益是否一样？

- 通过火焰图，可以看出**direct I/O**确实比buffer I/O**减少了kernel page的申请和拷贝动作**
- **buffer I/O**测试过程的中后期可以看到**kswapd**持续在高负载地**进行内存回收**，而direct I/O不需要







EROFS Direct I/O 8线程FIO测试对比（低内存场景）

- direct I/O和buffer I/O 8线程的fio测试（低内存场景，读取2.5GB文件）结果如下，提升达54.6%
- 对比普通场景和低内存场景的测试结果：**direct I/O性能比buffer I/O更稳定**
  - buffer I/O需要额外分配kernel page，而direct I/O不需要

	buffer I/O (MiB/s)	direct I/O (MiB/s)	提升比例
低内存场景	2350.0	3633.9	**54.6%**
普通场景	2629.8	3648.7	38.7%
降低比例	**10.6%**	**0.4%**	

总结

综合前述：

- EROFS direct I/O 实现符合预期，在大文件加载场景有如下优势：
  - 消除了额外的内存拷贝和双重内存占用，性能最高提升54.6%，并且性能较稳定
  - 避免系统因为双重缓存占用造成内存回收开销

- **开源提交：**
  - https://lore.kernel.org/linux-erofs/20250922124304.489419-1-guochunhai@vivo.com/T/#u

- **下一步计划：**
  - 支持 large folio
  - 支持异步 direct I/O

THANKS