Improving sparse file handling

API ideas for easier management of virtual disk images

LINUX PLUMBERS CONFERENCE

AUGUST 29-31, 2012 SAN DIEGO, CALIFORNIA

libvirt

VIRTUALIZATION API

Eric Blake <eblake@redhat.com>

29 August 2012

Talk Overview

- Introduction & Background
- Detection (xstat)
- Reading (lseek)
- Trimming (fallocate)
- Copying (posix_fadvise)
- Miscellaneous ideas

Introduction & Background

About this presentation

- User space perspective – what APIs can we add or improve to make sparse file and disk image management easier
- Needs feedback from kernel and file system developers to determine which features are feasible and worth pursuing, and in what timeframe
- Goal of efficiency – existing code already works as fallback mode, albeit slower

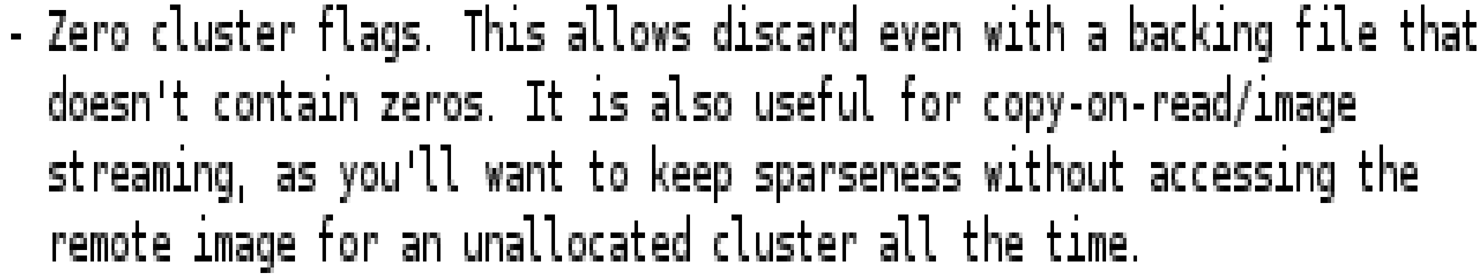
Sparse Files

- Modern file systems track 'holes', or large aligned portions containing only zero bytes, for less disk usage
  - Unallocated hole: Been around for years, created by seeking past EOF then writing
  - Allocated but unwritten hole: Newer, created by using [posix_]fallocate
  - Punching holes: ability to create either type of hole after file already exists

Virtual Machine Images

- Virtual machine images are typically sparse, allocated in the host only as the guest actually touches sectors
- virt-sparsify (libguestfs) exists to create sparse copy of a guest image
  - http://libguestfs.org/virt-sparsify.1.html

Virtual Machine Images

- New qcow2 version 3 file format, Apr '12
  - Metadata for marking an extent as sparse, and ability to discard sectors:
  - http://git.qemu.org/?p=qemu.git;h=4fabffc1
  - 

    - Zero cluster flags. This allows discard even with a backing file that doesn't contain zeros. It is also useful for copy-on-read/image streaming, as you'll want to keep sparseness without accessing the remote image for an unallocated cluster all the time.

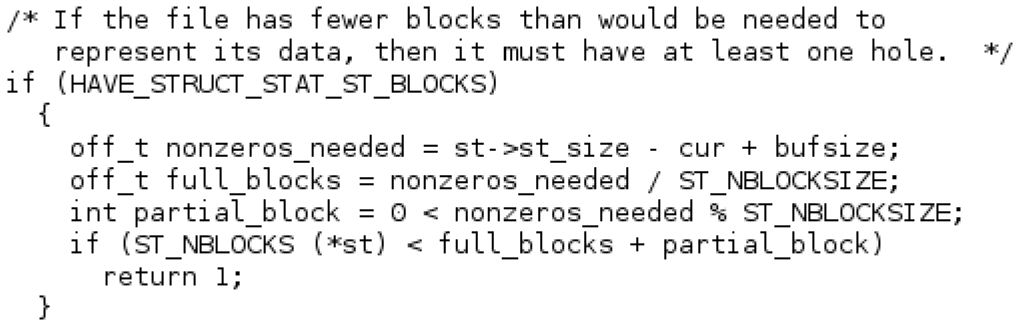
Trade-offs

- Sparse files allow disk over-commit
  - With that comes the risk of fragmentation and ENOSPC – modifying a previously unallocated sector requires time-consuming allocation
- Creation of fully-allocated images via [posix_]fallocate(), but desirable to still get same performance regarding read behavior of sparse files
  - Importance of effects on write() vs. read()

Detecting sparse files

Possibilities with xstat()

Case study: grep

- Detecting a sparse file up front allows time- and memory-saving decision
  - grep intentionally outputs “Binary file matches” for any file containing NUL
  - All sparse files contain NUL
- Traditional behavior, using only [f]stat()
  - http://git.sv.gnu.org/cgit/grep.git/tree/src/main.c?id=e1305800#n475
  - 

```
/* If the file has fewer blocks than would be needed to
   represent its data, then it must have at least one hole.  */
if (HAVE_STRUCT_STAT_ST_BLOCKS)
  {
    off_t nonzeros_needed = st->st_size - cur + bufsize;
    off_t full_blocks = nonzeros_needed / ST_NBLOCKSIZE;
    int partial_block = 0 < nonzeros_needed % ST_NBLOCKSIZE;
    if (ST_NBLOCKS (*st) < full_blocks + partial_block)
      return 1;
  }
```

Case study: grep

- Traditional approach fails for file systems that store small files in directory listing
    - https://lists.gnu.org/archive/html/bug-grep/2012-07/msg00018.html
- Also, with some file systems, compressed files can occupy fewer disk sectors than reported file size, even if not sparse
- Misses allocated but unwritten holes
- Can we design a faster, reliable way to detect that a file is sparse, including a way without open()ing it first?

xstat() history

- Version 6 patch in July 2010
  - http://thread.gmane.org/gmane.linux.kernel.cifs/225/focus=49713
- Why did xstat() stall?
  - https://lkml.org/lkml/2010/7/19/103
  - btime semantics, interface needs help
- Should this be revived? If so, can we add a field to answer whether a file is sparse?
- How expensive is sparse detection? Is a yes/no answer better than a count of sparse blocks?

Reading sparse files

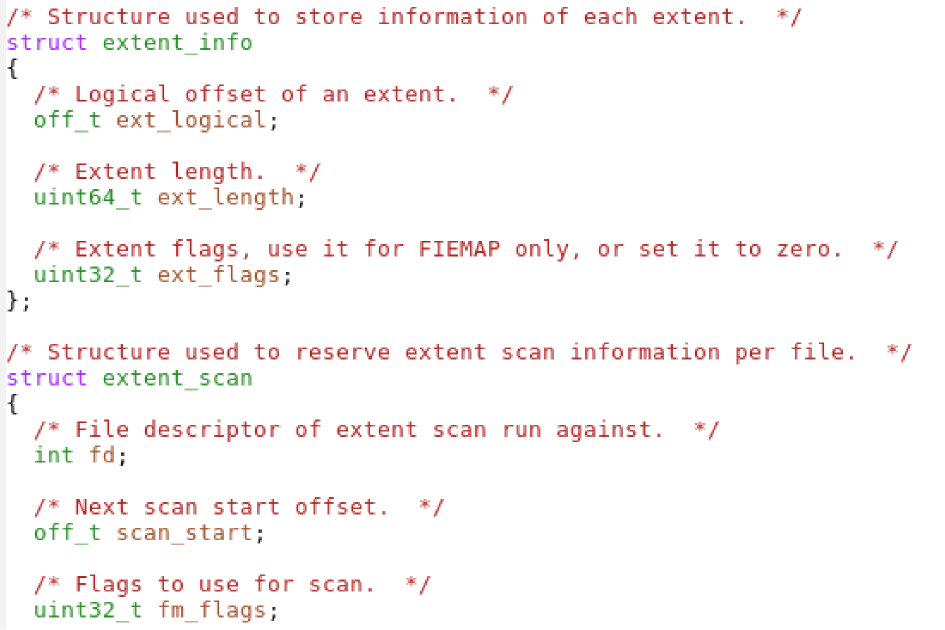
Possibilities with lseek() and SEEK_HOLE

Case Study: cp

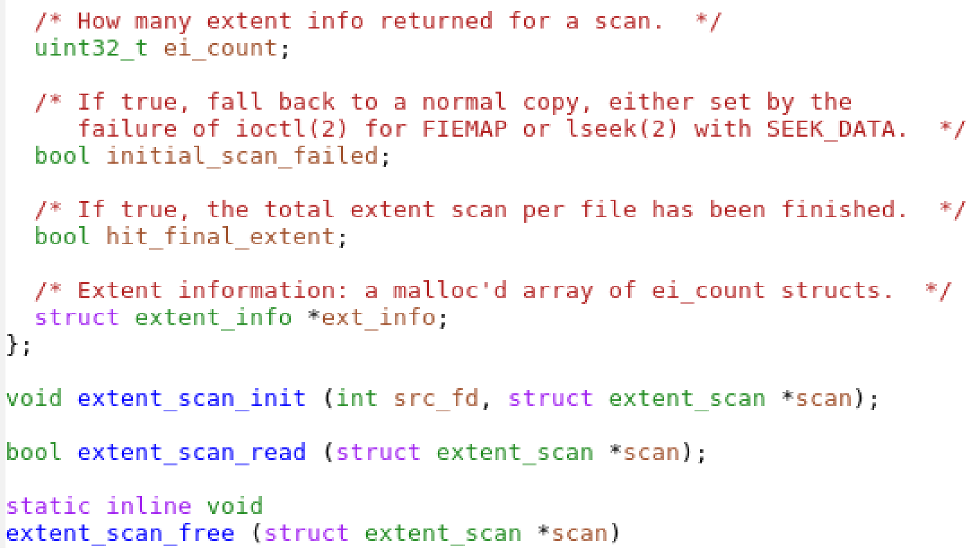
- GNU Coreutils 'cp --sparse' since Dec '95
  - But previously it unconditionally created sparse files, since before '92 initial commit
  - Brute force – read each sector in full, before skipping while writing the copy
- Solaris introduced SEEK_HOLE in '05
  - https://blogs.oracle.com/bonwick/entry/seek_hole_and_seek_data
- Later, Linux added ioctl(FIEMAP), Oct '07
  - https://lwn.net/Articles/260803/

Case Study: cp

- coreutils 8.10, Feb '11, started using SEEK_HOLE/FIEMAP for better cp, via a wrapper function, extent-scan.h
- http://git.sv.gnu.org/cgit/coreutils.git/tree/src/extent-scan.h#n35



```
/* Structure used to store information of each extent.  */
struct extent_info
{
  /* Logical offset of an extent.  */
  off_t ext_logical;

  /* Extent length.  */
  uint64_t ext_length;

  /* Extent flags, use it for FIEMAP only, or set it to zero.  */
  uint32_t ext_flags;
};

/* Structure used to reserve extent scan information per file.  */
struct extent_scan
{
  /* File descriptor of extent scan run against.  */
  int fd;

  /* Next scan start offset.  */
  off_t scan_start;

  /* Flags to use for scan.  */
  uint32_t fm_flags;
```



```
  /* How many extent info returned for a scan.  */
  uint32_t ei_count;

  /* If true, fall back to a normal copy, either set by the
     failure of ioctl(2) for FIEMAP or lseek(2) with SEEK_DATA.  */
  bool initial_scan_failed;

  /* If true, the total extent scan per file has been finished.  */
  bool hit_final_extent;

  /* Extent information: a malloc'd array of ei_count structs.  */
  struct extent_info *ext_info;
};

void extent_scan_init (int src_fd, struct extent_scan *scan);

bool extent_scan_read (struct extent_scan *scan);

static inline void
extent_scan_free (struct extent_scan *scan)
```

Case Study: cp

- SEEK_HOLE usage is simpler than FIEMAP, at the expense of fewer details
  - But reading only cares about locating holes, not whether the hole is allocated
- Uncovered some severe bugs in FIEMAP implementations, including the need to fsync() before information is reliable
  - Thankfully, most of these have been fixed
- gnulib considering adopting coreutils' hole iteration for use in other software

SEEK_HOLE usage

- Coreutils is an active user – great test bed for stressing new implementations
- Potential for other uses:
  - tar(1) optimizes output of sparse sectors
  - diff(1) and cmp(1) gain faster comparisons
  - rsync(1) can do faster transmission
  - qemu can process thin-provisioned images faster
  - More ideas?

SEEK_HOLE today

- Next POSIX version will add SEEK_HOLE
  - http://austingroupbugs.net/view.php?id=415
- Adoption into Linux began in Apr '11
  - https://lwn.net/Articles/440255/
  - Now present in BTRFS, XFS
  - Proposed for tmpfs, ext4, but missed 3.5.0
  - http://thread.gmane.org/gmane.linux.kernel.mm/82183/focus=65834
    - Chicken-and-egg – “But your vote would count for a lot more if you know of some app which would really benefit from this functionality in tmpfs: I've heard of none.” - Hugh Dickins

SEEK_HOLE improvements

- Road map for other file systems?
- lseek(SEEK_HOLE) changes file offset, as required in proposed POSIX wording
  - But libraries for multi-threaded programs use pread()/pwrite() to avoid changing offset behind back of another thread
  - Should we add new seek modes that return same location as SEEK_HOLE, but without modifying the file offset? What to name it?

SEEK_HOLE improvements

- Raw block devices also have holes
- 'GET LBA STATUS' on a SCSI disk can be used to track holes
  - https://lwn.net/Articles/355460/
- Being able to access this map through lseek(SEEK_HOLE), or even FIEMAP, would ease efforts
- Useful for partitions, LVM volumes, etc.

Trimming sparse files

Possibilities with fallocate()

Case Study: virt-sparsify

- Traditionally, holes could only be created at the end of a growing file
  - Once extent is allocated, can't reclaim space
- But the qcow2 virtual disk image format wants to create holes after the fact
- virt-sparsify uses copying to trim an offline disk image
  - Creation by copying is slow, and requires extra disk space

FALLOC_FL_PUNCH_HOLE

- Linux 2.6.38 added a flag to fallocate() FALLOC_FL_PUNCH_HOLE in Nov '10, to request creation of a hole in the file
  - https://lkml.org/lkml/2010/11/15/251
- SCSI distinguishes deallocate (drop extent allocation) from anchor (keep allocated, but treat as unwritten)
- Proposal to support both methods, by adding FALLOC_FL_ZERO_RANGE
  - https://lwn.net/Articles/501631/

fstrim impacts

- ATA TRIM command to inform block device of discarded extents
  - https://en.wikipedia.org/wiki/TRIM
  - But painfully slow when mounting -o discard
- Serial ATA 3.1 adding Queued Trim
  - This needs exposure through fallocate()
- Can user space request the difference between anchor and deallocate?

fstrim improvements

- Need solution across entire virt stack
  - Guest agent for host-initiated trim in guest
- For guests using SCSI passthrough and qemu 1.2, guests may send UNMAP
  - if using userspace iSCSI, it just works
  - otherwise, UNMAP and WRITE SAME commands require CAP_SYS_RAWIO
    - https://lkml.org/lkml/2012/7/20/273
- Future qemu will extend UNMAP support using fallocate() on local, network files

Copying sparse files

Possibilities with posix_fadvise()

Case study: libvirt saved image

- Libvirt can save guests across host reboot, using migration to disk
  - But doing multiple guests at once triggered fencing of the host from cache pollution
  - http://bugzilla.redhat.com/714752
- Libvirt implemented O_DIRECT code to bypass the problem, in Jul '11
  - http://libvirt.org/git/?p=libvirt.git;a=commit;h=1229165
- But using O_DIRECT has its limitations...

Case study: libvirt saved image

- Rather than having qemu directly write to fd, libvirt connects a pipe to a helper
- libvirt_iohelper must collect read()s from a pipe into a full buffer to then write() to the O_DIRECT fd, for more syscalls
  - Libvirt chose to only do aligned transfers; unaligned work is even more expensive with user-space read-modify-write
  - What is the cost of extra pipe I/O and context switching? Can kernel help?

posix_fadvise() overview

- O_DIRECT is not standard; the POSIX replacement appears, at first glance, to be posix_fadvise()
  - http://pubs.opengroup.org/onlinepubs/9699919799/functions/posix_fadvise.html
  - Libvirt's case would use these hints:
    - POSIX_FADV_SEQUENTIAL – will visit in order
    - POSIX_FADV_NOREUSE – no need to cache
- Needed in both directions – write on host shutdown, then read on host boot
  - neither pass should pollute file system cache

posix_fadvise() pitfalls

- Per POSIX: “The implementation may use this information to optimize handling of the specified data. The posix_fadvise() function shall have no effect on the semantics of other operations on the specified data, although it may affect the performance of other operations.”
- Oops – unlike O_DIRECT, this is advisory only, so kernel might not honor it

posix_fadvise() pitfalls

- Current Linux implementation:
  - POSIX_FADV_SEQUENTIAL merely doubles readahead window for read, but has no impact on write; is one-shot operation
  - POSIX_FADV_NOREUSE is currently a no-op (and before 2.6.18, it forced a preload as if by POSIX_FADV_WILLNEED)
- No flag to let application request that a particular access does not need caching

posix_fadvise() improvements

- Can posix_fadvise() be made stateful rather than one-shot, where a parent application can set flags on an fd, then pass it to a child process, and the flags are still in effect unless the child adds additional competing flags?
- Can we give feedback to the user when posix_fadvise hints are actually being honored? Would these hints live in procfs, in [f]pathconf(), or both?

posix_fadvise() improvements

- Can POSIX_FADV_NOREUSE drive the same benefits of file system cache avoidance of O_DIRECT, preferably without the painful overhead of mandating user-space alignment?
  - Kernel would have to use a bounce buffer for unaligned data, but coupled with a hint on sequential usage, would know soonest moment to take it back out of cache
- http://bugzilla.redhat.com/722185

Miscellaneous improvements

Related improvements

- Other storage-related requests from qemu, for efficiently accessing images
- https://lists.gnu.org/archive/html/qemu-devel/2012-07/msg04169.html
  - Support for connecting to an iSCSI target without scanning partitions
  - Support for fsync/fdatasync with ranges (or alternatively, sync_file_range that writes metadata)
- Support for fallocate() on block devices

copy-on-write improvements

- LVM, BTRFS, several NAS devices, and growing number of other storage solutions are coming up with independent copy-on-write solutions
- Can we come up with a common interface for driving a copy-on-write fork of file contents?

Summary

LPC 2012: Improving sparse file handling

libvirt

VIRTUALIZATION API

http://libvirt.org/