GEOM_VIRSTOR

Or: how to fool yourself into thinking that you don't have to worry about buying more hard drives

Ivan Voras <ivoras@freebsd.org>

Google Summer of Code project, 2006 ; Mentor: Pawel Jakub Dawidek <pjd@freebsd.org>

What is GEOM_VIRSTOR?

- Disk storage "overcommit" GEOM class
- Allows you to:
  - Create a huge virtual hard drive, backed by arbitrary number of small(er) hard drives
  - Add drives to the "virstor" device when you need them
- Usage:
  - Create a huge (multi-TB) file system on a 100 GB hard drive, add more hard drives when you need the space

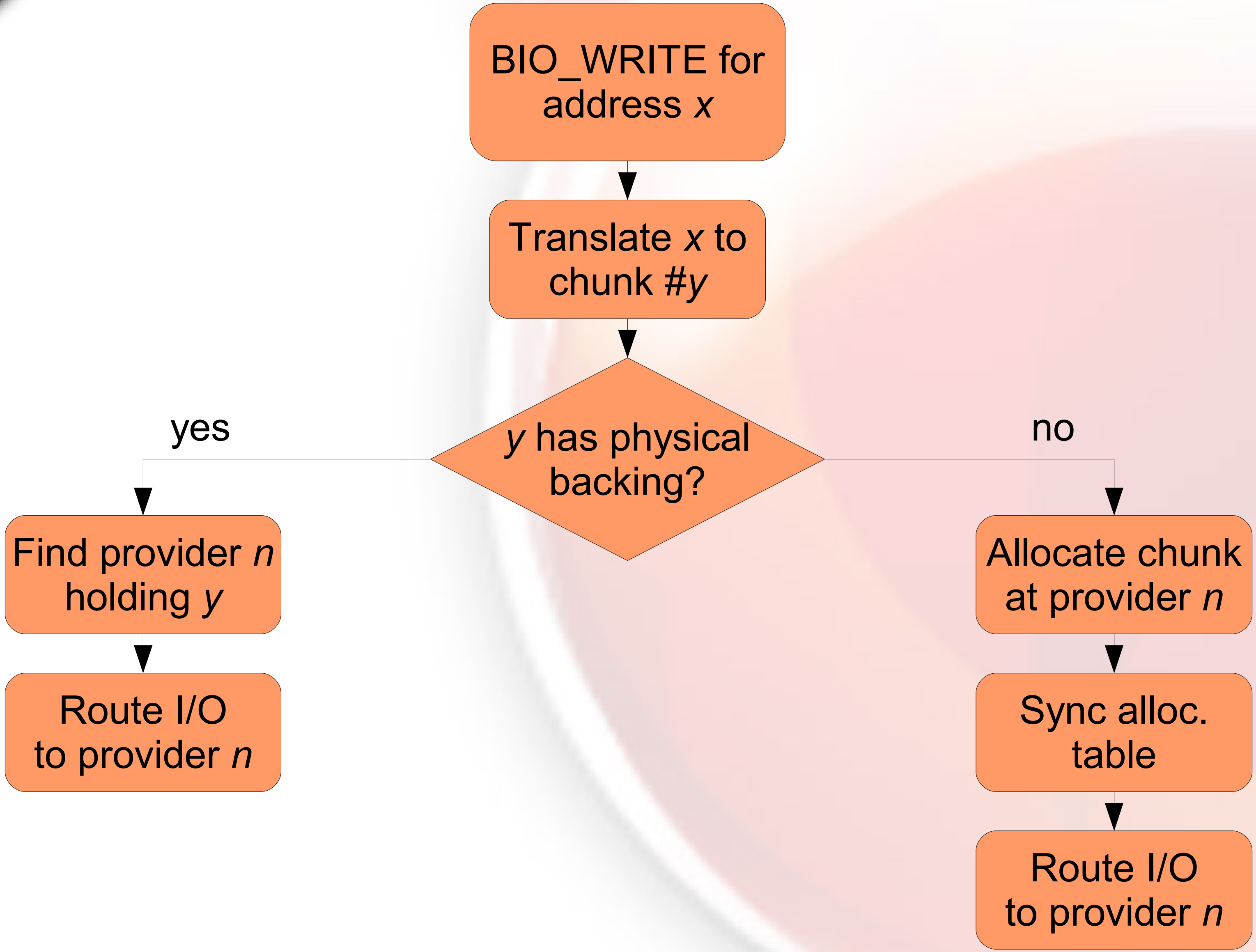
What's it for?

- The basic purpose is storage virtualization, literally :)
- **Avoids the need for `growfs`**
  - but there are consequences...
- It's generic, and usable with any file system (`ufs`, `msdosfs`, `ext2fs`)
  - Perversion: create a RAID array on top of it...
- It's been created before ZFS, which has a much nicer way to extend storage

How does it work?

- First step: label components to belong to gvirstor, specify virtual size
  - This will initialize the allocation table on the first component
  - All space divided into chunks (default size: 4 MB)
- Bringing the geom up will create a `/dev/virstor/foo` device of specified (virtual) size
- Writing `to /dev/virstor/foo` will sequentially allocate space from components

Operation (BIO_WRITE case)



BIO_WRITE for address *x*

Translate *x* to chunk #*y*

y has physical backing?

yes

Find provider *n* holding *y*

Route I/O to provider *n*

no

Allocate chunk at provider *n*

Sync alloc. table

Route I/O to provider *n*

The allocation table

- Holds one entry per *virtual* chunk

int16 flags	int16 prov_no	int32 chunk_no

· · ·

- Stored at the start of the first provider
  - (in a continuous set of chunks)
- Only BIO_WRITE requests can allocate new chunks (if needed)
  - BIO_READ on address without physical backing returns zeroes

Implementation

- Simple / non-threaded GEOM class
- Allocation table is always written synchronously
  - (i.e. before BIO_WRITE is marked as completed)
- Keeps track of chunk usage
  - Blocks BIO_WRITE requests if there's no physical storage (drives) available to allocate from
  - Notifies admin via kernel message
- Allows "hot" insertion of new drives
  - Hot removal also, but only if no blocks allocated and it's the last drive

Limitations

- **Applications that want to be "smart" about storing data sequentially on a drive are defeated**
  - Canonical example: UFS cylinder groups
  - The point of having `cgs` is for them to be spread across the (physical) drive, to "group" data
  - After `newfs` is done, all cylinder groups (superblock backups, inode & block tables) will (physically) be stored almost sequentially on the first drive in virstor
  - Big fragmentation problems

Ideas / future work

- Implement moving allocated chunks from one drive to other drives (to clear that drive from allocated data)
- Implement removing drives from the middle of the virstor set
- ... ?
- Current status: waiting to be committed to -CURRENT

The End

- Thanks:
  - To FreeBSD Foundation for funding the trip to BSDCan
  - To Google for sponsoring the project & part of the expenses here
  - Pawel & many other people for helping create gvirstor
- Questions?

Contact: Ivan Voras <ivoras@freebsd.org>
Project homepage: http://wiki.freebsd.org/gvirstor