Optimizing fs on sd-card for Linux/Fedora on Dreamplug

The write performance on the SD-card in my Dreamplug (sweet little Arm computer) was abysmally slow. Any write on large files would be ok but any writes that involved many small files would take forever. We are talking about 3-4kb/s for some yum operations.

Pre-measurements using very simple tests

1. Single large file write: 4.14mb/s (commands: "sync; rm testing; sync; time (dd if=/dev/zero of=testing bs=16k count=10000; sync)")
2. Many smallish files write: 366kb/s (commands: "sync; rm -rf testing*; sync; time (for item in `seq 1 1000`; do dd if=/dev/zero of=testing.$item bs=16k count=10; sync; done;)")

Problem

My SD-Card is a PNY SDHC 8gb Class 10 device that boasts 20 mb/s transfer. I had it formatted as ext2, as it that is widely advised as "good". It very simply does not perform, especially on writes, but reads are also not good.. Linux on the Dreamplug feels sluggish, yum is especially slow taking minutes to even install small packages.

Synopsis of getting this to perform:

1. Find page, erase block and segment sizes of the SD-card.
2. Format as ext4
   1. Without journalling – reduces IO and reduces the writes to the SD card ultimately reducing wear.
   2. Set ext4 raid parameters (stride and stripe-width) – ext4 will optimize IO and avoid read-erase-write cycles if possible
3. Set mount options to further reduce IO

Finding page, erase block and segment sizes

First of, I used a tool called FlashBench to determine the flash parameters of my SD-card as it did not appear in the handy list at https://wiki.linaro.org/WorkingGroups/Kernel/Projects/FlashCardSurvey?action=show&redirect=WorkingGroups%2FKernelConsolidation%2FProjects%2FFlashCardSurvey

After downloading the tool from `git://git.linaro.org/people/arnd/flashbench.git` (http://git.linaro.org/gitweb?p=people/arnd/flashbench.git;a=tree), compile and run:

```
[root@megan flashbench-HEAD-4fb06b5]# ./flashbench -a /dev/mmcblk0
 align 2147483648        pre 534µs       on 661µs        post 546µs
 align 1073741824        pre 570µs       on 719µs        post 563µs
 align 536870912 pre 551µs       on 700µs        post 547µs      di
 align 268435456 pre 574µs       on 708µs        post 569µs      di
 align 134217728 pre 548µs       on 676µs        post 542µs      di
 align 67108864  pre 555µs       on 681µs        post 541µs      di
 align 33554432  pre 548µs       on 691µs        post 548µs      di
 align 16777216  pre 550µs       on 680µs        post 542µs      di
 align 8388608   pre 566µs       on 707µs        post 581µs      di
 align 4194304   pre 544µs       on 558µs        post 525µs      di
 align 2097152   pre 539µs       on 551µs        post 535µs      di
 align 1048576   pre 538µs       on 548µs        post 535µs      di
 align 524288    pre 537µs       on 545µs        post 535µs      di
 align 262144    pre 538µs       on 548µs        post 536µs      di
 align 131072    pre 536µs       on 547µs        post 532µs      di
 align 65536     pre 536µs       on 547µs        post 533µs      di
 align 32768     pre 536µs       on 548µs        post 533µs      di
 align 16384     pre 536µs       on 549µs        post 535µs      di
 align 8192      pre 512µs       on 555µs        post 546µs      di
 align 4096      pre 555µs       on 562µs        post 563µs      di
 align 2048      pre 537µs       on 538µs        post 534µs      di
```

Analysis:

1. Interesting parts of this result are the diff changes drastically at two places:
    1. from 8388608 (8Mb) to 4194304 (4MB): Based in example readme in flashbench, this indicates that there was no performance overhead reading two blocks over the 4mb boundary, but there was for 8mb boundary. The guess is then that the erasure block is 8mb large on my sd-card
    2. before 8192 and after. I would really like to know why there is a bump at 8k, but times after that are so much lower, so 8k is obviously some sort of boundary point.
2. From this, I deduce two things,
    1. Ext4 should have a block size of 4k, and the “stride” value should be 2. This will cause ext4 to think that units of 2 blocks (8k) can and should be treated as one.
    2. Ext4 should have the stripe-size set to 1024. This value was calculated by taking 8M (guessed erasure block size) dividing by 8K (size of a stride, 2 times block size (4K)). This will (hopefully) cause Ext4 to try to align writes so that while erasure blocks are written continuously and make it avoid sub-block updates.

Repartition and reformat of the sd-card with new settings

Note that I start the partition on an erasure block, i.e. at 8mb into the SD card, to make absolutely sure that everything becomes aligned. Fdisk uses blocks of 512 bytes, so that means that we want to start at 8*1024^2/512 = 16384.

```
[root@megan ~]# fdisk /dev/mmcblk0
Command (m for help): n
 Partition type:
 p   primary (0 primary, 0 extended, 4 free)
 e   extended
 Select (default p):
```

```
 Using default response p
 Partition number (1-4, default 1):
 Using default value 1
 First sector (2048-15759359, default 2048): 16384
 Last sector, +sectors or +size{K,M,G} (16384-15759359, default 157!
 Using default value 15759359
Command (m for help): p
Disk /dev/mmcblk0: 8068 MB, 8068792320 bytes
 4 heads, 16 sectors/track, 246240 cylinders, total 15759360 sector:
 Units = sectors of 1 * 512 = 512 bytes
 Sector size (logical/physical): 512 bytes / 512 bytes
 I/O size (minimum/optimal): 512 bytes / 512 bytes
 Disk identifier: 0x00000000
Device Boot      Start         End      Blocks   Id  System
 /dev/mmcblk0p1           16384    15759359     7871488   83  Linux
Command (m for help): w
 The partition table has been altered!
Calling ioctl() to re-read partition table.
 Syncing disks.
 [root@megan ~]#
```

Reformat the filesystem, this time with Ext4 with block size of 4k, without journaling, but with additional parameters to encourage Ext4 to do the right thing with respect to the erasure block:

```
[root@megan ~]# mkfs.ext4 -O ^has_journal -E stride=2,stripe-width=:
 mke2fs 1.41.14 (22-Dec-2010)
 Filesystem label=Fedora14Arm
 OS type: Linux
 Block size=4096 (log=2)
 Fragment size=4096 (log=2)
 Stride=2 blocks, Stripe width=1024 blocks
 492880 inodes, 1967872 blocks
 98393 blocks (5.00%) reserved for the super user
 First data block=0
 Maximum filesystem blocks=2017460224
 61 block groups
 32768 blocks per group, 32768 fragments per group
 8080 inodes per group
 Superblock backups stored on blocks:
 32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632
```

```
Writing inode tables: done
 Writing superblocks and filesystem accounting information: done
This filesystem will be automatically checked every 28 mounts or
 180 days, whichever comes first.  Use tune2fs -c or -i to override
[root@megan ~]#
```

Changes to mount

To further reduce IO and increase performance, the following seem to be common recommendations, that I did.

1. Change /etc/fstab so that entry for root partition fourth parameter is changed from “default” to “default,noatime,nodiratime”
2. Add linux boot parameter “elevator=noop” in u-boot so that the kernel does not assume that the disk is a spinning medium and does not try to incorrectly optimize the IO.

Simple performance test after all the changes

1. Single large file write: 8.2mb/s (commands: “sync; rm testing; sync; time (dd if=/dev/zero of=testing bs=16k count=10000; sync)”)
2. Many small-ish files write: 2.7mb/s (commands: “sync; rm -rf testing*; sync; time (for item in `seq 1 1000`; do dd if=/dev/zero of=testing.$item bs=16k count=10; sync; done;)”)

RESULT: nearly 10 times performance increase for small-ish files!!!!

Sources:

http://lwn.net/Articles/428584/
http://www.raspberrypi.org/forum/projects-and-collaboration-general/optimizing-linux-for-flash-memory

This entry was posted on 2012/01/14 at 5:55 pm and is filed under Linux. You can follow any responses to this entry through the RSS 2.0 feed. You can leave a response, or trackback from your own site.