Impact of space usage on file system performance

Introduction

This page is about testing the effect of file system usage on performance. This project was inspired by Linux Questions thread "How is the minimum file system free space decided?".

Common advice is to limit file system usage to ~75 or ~80%. Is there any test data behind that recommendation? The recommendation is old, dating from when a 1 GB file system was huge; is this limit still applicable now 3 TB HDDs are commodity items?

I could not find any such test data. The nearest was file system tests -- usually comparing the performance of various file system types or various hardware platforms in simulated "real world" scenarios -- but none were found that focussed on the effects of file system space usage.

There may be non-performance reasons for keeping 20% or more free space on a file system. As acid_kewpie pointed out in the Linux Questions thread, free space is useful to accommodate unexpected data growth.

Initial results

06-Aug-2013 filesystem_freespace_test results: 15 GB ext4 on SATA 6 port

The attached .tgz has the test script log, the test report, gnuplot scripts and screen shots of the gnuplots. The test report includes some figures which are not plotted including the normalised standard deviation for each set of four tests.

After unpacking, the plots may be generated using this bash command:

```
for i in {0..9}; do gnuplot --persist 06-Aug-2013\@09\:26.*.gnuplot_script.$i; done
```

The results do not tell a clear story. Of the ten plots:

- Three show the conventional result of task times getting longer as file system usage increases but the effect starts around 50 to 60%, not the conventional 75 to 80%. All three tests concern 10,000 directories (creating them, finding them and finding and removing them).
- Five show task times staying broadly similar throughout the test, with file system usage changing from 30 to 98%. All five show the test task taking longer during roughly 60 to 80% usage and returning to pre-60% times until the end of the test at 98% usage. These five tests concern either 10,000 directories (creating them, finding them and finding and removing them) or a 0.5 GiB file (reading or creating it).
- The remaining two plots show test times more or less consistent from 30 to 80% file system usage and then reducing by more than 50% from 80 to 98% usage.

The test script

https://bitbucket.org/charles_atkinson/public/commits/3366a2f75ef654d020c7fcdcf118b745cb8a2983

From the script's header comment, describing its purpose and actions:

```
 * Tests performance of file system at increasing levels of space usage.
   Notes:
   1. "Space usage" is what df lists as Use%.
   2.  Use% is "used/(used+free)", not the intuitively obvious "used/size".
 * Tests the performance of the underlying device, regardless of the file
   system (hdparm -tT).
   This is intended to give a baseline read-from-device figure, bypassing
   the Linux file system cache for comparison with later read tests.
 * Writes files until usage exceeds a configurable threshold by not more
   than 1%.
   This is intended to simulate typical usage.
   Notes:
   1. The files are filled with zeroes and have sizes randomly chosen
```

> *from a configurable range.*
>
> *2. The random size choice is weighted to create more smaller files than larger ones, using a e^(k*(x-1)) function where k is a configurable constant and x is a random number between 0 and 1. Thanks to Kaushik for the weighting function.*
>
> ** For every ten files created:*
>
> *1. The file system's and device's buffers are flushed (hdparm -f -F).*
>
> *2. A file chosen at random is deleted.*
>
> *3. The file system's and device's buffers are flushed again.*
>
> *This is intended to simulate typical usage and to produce fragmentation.*
>
> ** When the usage exceeds the configurable threshold, timed performance tests are run, as initialised in the initialise function, and test results are written to the report file. The report file is intended for use by gnuplot. Data is subsequently parsed out of its comment lines by the write_gnuplot_scripts function.*
>
> ** When the tests are complete increment the file system usage by a configurable percentage unless the configurable maximum percentage would be exeeded. If the usage is incremented, repeat the tests.*
>
> ** Write a gnuplot script file for each test*

The script attempts to run each test under identical conditions, avoiding buffering effects. Before each test is run this function is run:

```
#--------------------------
# Name: flush_buffers
# Purpose: flushes the OS file system buffer and, if the HDD implements it,
#   the HDD's write cache buffer
#--------------------------
function flush_buffers {
    #fct "${FUNCNAME[0]}" 'started'
    local buf

    ((EUID!=0)) && return    # Need to be root to flush buffers

    # Sync and flush the buffer cache
    # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    # Use hdparm -f rather than blockdev --flushbufs because it runs the same
    # flushing system call plus five others.
    buf=$(hdparm -f "$hdd_dev" 2>&1)
    rc=$?
    ((rc>0)) && msg E "Unable to flush the buffer cache: $buf"

    # Free cached dentries and inodes
    # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    # Required in case the above keeps a copy in memory which reads could use.
    # Normally preceded by calling sync but hdparm -f calls sync
    # References:
    # * https://www.kernel.org/doc/Documentation/sysctl/vm.txt
    # * http://catalin-festila.blogspot.in/2011/06/myth-of-dropcaches.html
    echo 2 > /proc/sys/vm/drop_caches

    # Flush the on-drive write cache buffer
    # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    buf=$(hdparm -F "$hdd_dev" 2>&1)
    rc=$?
    ((rc>0)) && msg E "Unable to flush the on-drive write cache buffer: $buf"

    #msg D "${FUNCNAME[0]}: Key /proc/meminfo values: $(grep -E '^Cached:|^Dirty:' /proc/meminfo)"

    #fct "${FUNCNAME[0]}" 'returning'
}  # end of function flush_buffers
```

`strace` shows that `hdparm -f` runs `sync(), fsync(3), fdatasync(3), sync(), ioctl(3, BLKFLSBUF, 0)` and finally `sync()`again. Despite that and – more importantly for read tests – `echo 2 > /proc/sys/vm/drop_caches`, there still appear to be buffering effects in the "Read 0.5 GiB file" test. Here's an illustrative line from the test log:

> *Raw results: 4.241 0.133 0.109 0.308. Result (mean): 1.2. Standard deviation of normalised results: 1.5*