Linux bcc ext4 Latency Tracing

06 Oct 2016

Storage I/O performance issues are often studied at the block device layer, but instrumenting the file system instead provides more relevant metrics for understanding how applications are affected.

My ext4dist tool does this for the ext4 file system, and traces reads, writes, opens, and fsyncs, and summarizes their latency as a power-of-2 histogram. For example:

```
# ext4dist
Tracing ext4 operation latency... Hit Ctrl-C to end.
^C

operation = 'read'
     usecs               : count     distribution
         0 -> 1          : 1210      |****************************************|
         2 -> 3          : 126       |****                                    |
         4 -> 7          : 376       |************                            |
         8 -> 15         : 86        |**                                      |
        16 -> 31         : 9         |                                        |
        32 -> 63         : 47        |*                                       |
        64 -> 127        : 6         |                                        |
       128 -> 255        : 24        |                                        |
       256 -> 511        : 137       |****                                    |
       512 -> 1023       : 66        |**                                      |
      1024 -> 2047       : 13        |                                        |
      2048 -> 4095       : 7         |                                        |
      4096 -> 8191       : 13        |                                        |
      8192 -> 16383      : 3         |                                        |

operation = 'write'
     usecs               : count     distribution
         0 -> 1          : 0         |                                        |
         2 -> 3          : 0         |                                        |
         4 -> 7          : 0         |                                        |
         8 -> 15         : 75        |****************************************|
        16 -> 31         : 5         |**                                      |

operation = 'open'
     usecs               : count     distribution
         0 -> 1          : 1278      |****************************************|
         2 -> 3          : 40        |*                                       |
         4 -> 7          : 4         |                                        |
         8 -> 15         : 1         |                                        |
        16 -> 31         : 1         |                                        |
```

This output shows a bi-modal distribution for read latency, with a faster mode of less than 7 microseconds, and a slower mode of between 256 and 1023 microseconds. The count column shows how many events fell into that latency range. It's likely that the faster mode was a hit from the in-memory file system cache, and the slower mode is a read from a storage device (disk).

This "latency" is measured from when the operation was issued from the VFS interface to the file system, to when it completed. This spans everything: block device I/O (disk I/O), file system CPU cycles, file system locks, run queue latency, etc. This is a better measure of the latency suffered by applications reading from the file system than measuring this down at the block device interface. Measuring at the block device level is better suited for other uses: resource capacity planning.

Note that this tool only traces the common file system operations previously listed: other file system operations (eg, inode operations including getattr()) are not traced.

ext4dist is a bcc tool that uses kernel dynamic tracing (via kprobes) with BPF. bcc is a front-end and a collection of tools that use new Linux enhanced BPF tracing capabilities.

I also wrote ext4slower, to trace these ext4 operations that are slower than a custom threshold. Eg, 1 millisecond:

```
# ext4slower 1
Tracing ext4 operations slower than 1 ms
TIME     COMM           PID    T BYTES   OFF_KB   LAT(ms) FILENAME
06:49:17 bash           3616   R 128     0           7.75 cksum
06:49:17 cksum          3616   R 39552   0           1.34 [
06:49:17 cksum          3616   R 96      0           5.36 2to3-2.7
06:49:17 cksum          3616   R 96      0          14.94 2to3-3.4
06:49:17 cksum          3616   R 10320   0           6.82 411toppm
06:49:17 cksum          3616   R 65536   0           4.01 a2p
06:49:17 cksum          3616   R 55400   0           8.77 ab
06:49:17 cksum          3616   R 36792   0          16.34 aclocal-1.14
06:49:17 cksum          3616   R 15008   0          19.31 acpi_listen
06:49:17 cksum          3616   R 6123    0          17.23 add-apt-repository
06:49:17 cksum          3616   R 6280    0          18.40 addpart
06:49:17 cksum          3616   R 27696   0           2.16 addr2line
06:49:17 cksum          3616   R 58080   0          10.11 ag
06:49:17 cksum          3616   R 906     0           6.30 ec2-meta-data
06:49:17 cksum          3616   R 6320    0          10.00 animate.im6
[...]
```

This is great for proving or exonerating the storage subsystem (file systems, volume managers, and disks) as a source of high latency events. Let's say you had occasional 100 ms application request outliers, and suspected it may be a single slow I/O (of 100 ms) or several (adding to 100 ms). Running "ext4slower 10" would print everything beyond 10 ms, proving or exonerating these theories. (Note that it could be thousands of sub-1 ms I/O, not caught by a "ext4slower 10".)

The output will be far too verbose, but you can also use a threshold of "0" to dump all events:

```
# ext4slower 0
Tracing ext4 operations
TIME     COMM           PID    T BYTES   OFF_KB   LAT(ms) FILENAME
06:58:05 supervise      1884   O 0       0           0.00 status.new
06:58:05 supervise      1884   W 18      0           0.02 status.new
06:58:05 supervise      1884   O 0       0           0.00 status.new
06:58:05 supervise      1884   W 18      0           0.01 status.new
06:58:05 supervise      15817  O 0       0           0.00 run
06:58:05 supervise      15817  R 92      0           0.00 run
06:58:05 supervise      15817  O 0       0           0.00 bash
06:58:05 supervise      15817  R 128     0           0.00 bash
06:58:05 supervise      15817  R 504     0           0.00 bash
06:58:05 supervise      15817  R 28      0           0.00 bash
06:58:05 supervise      15817  O 0       0           0.00 ld-2.19.so
[...]
```

Do run ext4dist first, to check the rate of file system operations. If it's millions per second, then running "ext4slower 0" will try to print millions of lines of output per second – probably not what you want, and will cost overhead on the system.

These tools are in bcc, and which has man pages (under /man/man8) and text files with more example screenshots (under /tools/*_example.txt). Each tool also has a USAGE message, eg:

```
# ext4slower -h
usage: ext4slower [-h] [-j] [-p PID] [min_ms]

Trace common ext4 file operations slower than a threshold

positional arguments:
  min_ms             minimum I/O duration to trace, in ms (default 10)

optional arguments:
  -h, --help         show this help message and exit
  -j, --csv          just print fields: comma-separated values
  -p PID, --pid PID  trace this PID only

examples:
    ./ext4slower             # trace operations slower than 10 ms (default)
    ./ext4slower 1           # trace operations slower than 1 ms
    ./ext4slower -j 1        # ... 1 ms, parsable output (csv)
    ./ext4slower 0           # trace all operations (warning: verbose)
    ./ext4slower -p 185      # trace PID 185 only
```

There are also equivalent tools in bcc for btrfs, xfs, and zfs (so far).

You can comment here, but I can't guarantee your comment will remain here forever: I might switch comment systems at some point (eg, if disqus add advertisements).

Copyright 2017 Brendan Gregg.
About this blog