Block I/O Layer Tracing: *blktrace*

Gelato – Cupertino, CA
April 2006

Alan D. Brunelle

Hewlett-Packard Company
Open Source and Linux Organization
Scalability & Performance Group

Alan.Brunelle@hp.com

Introduction

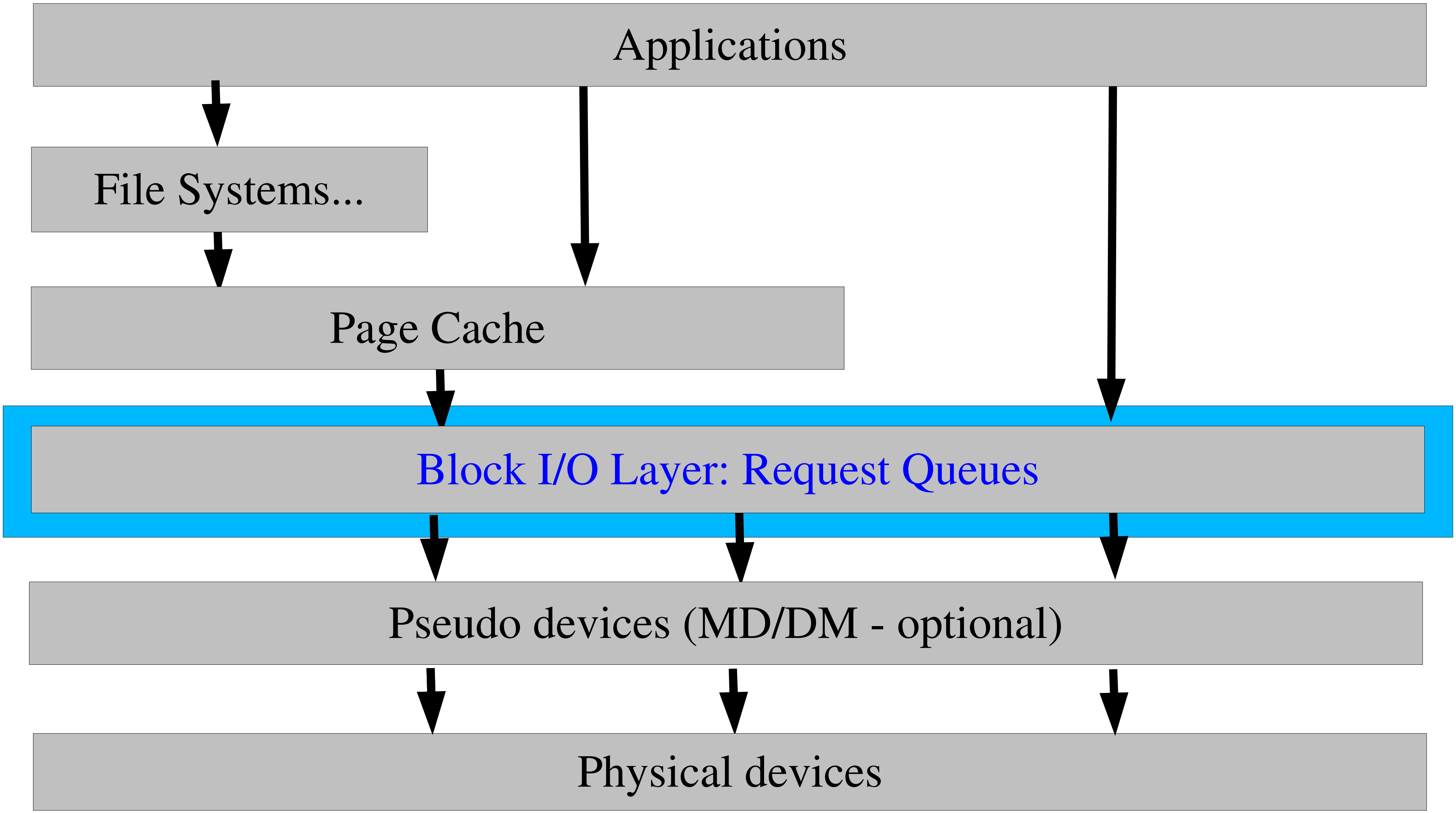
- Blktrace – overview of a new Linux capability
  - Ability to see what's going on inside the block I/O layer
    - “You can't count what you can't measure”
  - Kernel implementation
  - Description of user applications
- Sample Output & Analysis

Problem Statement

- Need to know the specific operations performed upon ***each*** I/O submitted into the block I/O layer

- Who?
  - Kernel developers in the I/O path:
    - Block I/O layer, I/O scheduler, software RAID, file system, ...
  - Performance analysis engineers – *HP OSLO S&P*...

Block I/O Layer (simplified)



iostat

- The iostat utility *does* provide information pertaining to request queues associated with specifics devices
  - Average I/O time on queue, number of merges, number of blocks read/written, ...
- However, it does *not* provide detailed information on a per-I/O basis

Blktrace – to the rescue!

- Developed and maintained by Jens Axboe (block I/O layer maintainer)
  - My additions included adding threads & utility splitting, DM remap events, blkrawverify utility, binary dump feature, testing, kernel/utility patches, and documentation.
- Low-overhead, configurable kernel component which emits ***events*** for specific operations performed on each I/O entering the block I/O layer
- Set of tools which extract and format these events

However, *blktrace* is **<u>not</u>** an analysis tool!

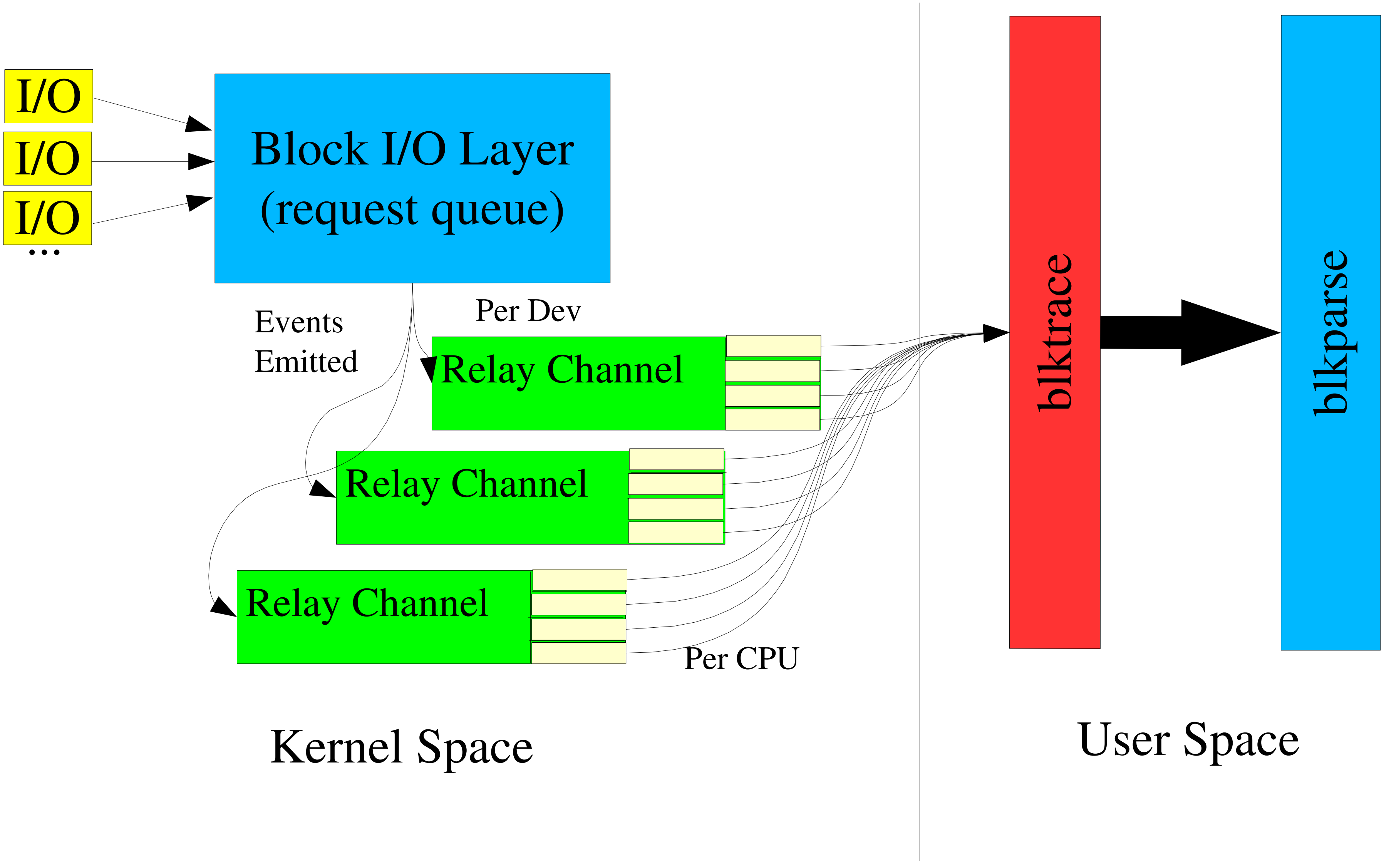
Feature List

- Provides detailed block layer information concerning individual I/Os
- Low-overhead kernel tracing mechanism
  - Seeing less than 2% hits to application performance in relatively stressful I/O situations
- Configurable:
  - Specify 1 or more physical or logical devices (including MD and DM (LVM2))
  - User-selectable events – can specify filter at event acquisition and/or when formatting output
- Supports both "live" and "playback" tracing

Events Captured

- Request queue entry allocated
- Sleep during request queue allocation
- Request queue insertion
- Front/back merge of I/O on request queue
- Re-queue of a request
- Request issued to underlying block dev
- Request queue plug/unplug op
- I/O split/bounce operation
- I/O remap
  - MD or DM
- Request completed

blktrace: General Architecture



blktrace Utilities

- *blktrace:* Device configuration, and event extraction utility
  - Store events in (long) term storage
  - Or, pipe to *blkparse* utility for live tracing
    - Also: networking feature to remote events for parsing on another machine
- *blkparse:* Event formatting utility
  - Supports textual or binary dump output

blktrace: Event Output

```
% blktrace -d /dev/sda -o - | blkparse -i -
8,0 3  1 0.000000000 697 G W 223490 + 8 [kjournald]
8,0 3  2 0.000001829 697 P R [kjournald]
8,0 3  3 0.000002197 697 Q W 223490 + 8 [kjournald]
8,0 3  4 0.000005533 697 M W 223498 + 8 [kjournald]
8,0 3  5 0.000008607 697 M W 223506 + 8 [kjournald]
...
8,0 3 10 0.000024062 697 D W 223490 + 56 [kjournald]
8,0 1 11 0.009507758   0 C W 223490 + 56 [0]
```

Dev <mjr, mnr>

CPU

Process

Sequence Number

Time Stamp

PID

Event

Start block + number of blocks

blktrace: Summary Output

Per CPU details

Writes submitted on

```
CPU0 (sdao):
 Reads Queued:           0,        0KiB  Writes Queued:      77,382,    5,865MiB
 Read Dispatches:        0,        0KiB  Write Dispatches:    7,329,    3,020MiB
 Reads Requeued:         0               Writes Requeued:         6
 Reads Completed:        0,        0KiB  Writes Completed:        0,        0KiB
 Read Merges:            0               Write Merges:       68,844
 Read depth:             2               Write depth:            65
 IO unplugs:           414               Timer unplugs:         414
...
CPU3 (sdao):
 Reads Queued:         105,       18KiB  Writes Queued:      14,541,    2,578MiB
 Read Dispatches:       22,       60KiB  Write Dispatches:    6,207,    1,964MiB
 Reads Requeued:         0               Writes Requeued:     1,408
 Reads Completed:       22,       60KiB  Writes Completed:   12,300,    5,059MiB
 Read Merges:           83               Write Merges:       10,968
 Read depth:             2               Write depth:            65
 IO unplugs:           287               Timer unplugs:         287

Total (sdao):
 Reads Queued:         105,       18KiB  Writes Queued:      92,546,    8,579MiB
 Read Dispatches:       22,       60KiB  Write Dispatches:   13,714,    5,059MiB
 Reads Requeued:         0               Writes Requeued:     1,414
 Reads Completed:       22,       60KiB  Writes Completed:   12,300,    5,059MiB
 Read Merges:           83               Write Merges:       80,246
 IO unplugs:           718               Timer unplugs:         718

Throughput (R/W): 0KiB/s / 39,806KiB/s
Events (sdao): 324,011 entries
Skips: 0 forward (0 -   0.0%)
```

Writes completed on

Per device details

Avg throughput

blktrace: Event Storage Choices

- Physical disk backed file system
  - *Pros*: large/permanent amount of storage available; supported by all kernels
  - *Cons*: potentially higher system impact; may negatively impact devices being watched (if storing on the same bus that other devices are being watched on...)
- RAM disk backed file system
  - *Pros*: predictable system impact (RAM allocated at boot); less impact to I/O subsystem
  - *Cons*: limited/temporary storage size; removes RAM from system (even when not tracing); may require reboot/kernel build
- TMPFS
  - *Pros*: less impact to I/O subsystem; included in most kernels; only utilizes system RAM while events are stored
  - *Cons*: limited/temporary storage; impacts system predictability – RAM “removed” as events are stored – could affect application being “watched”

blktrace: Analysis *Aid*

- As noted previously, *blktrace* does <u>not</u> analyze the data; it is responsible for storing and formatting events
- Need to develop post-processing analysis tools
  - Can work on formatted output or binary data stored by *blktrace* itself
  - Example: *btt* – block trace timeline

Practical *blktrace*

- Here at HP OSLO S&P, we are investigating I/O scalability at various levels
  - Including the efficiency of various hardware configurations and the effects on I/O performance caused by software RAID (MD and DM)
- *blktrace* enables us to determine scalability issues within the block I/O layer and the overhead costs induced when utilizing software RAID

Life of an I/O *(simplified)*

- I/O enters block layer – it can be:
  - Remapped onto another device (MD, DM)
  - Split into 2 separate I/Os (alignment, size, ...)
  - Added to the request queue
  - Merged with a previous entry on the queue

    All I/Os end up on a request queue at some point
- At some later time, the I/O is issued to a device driver, and submitted to a device
- Later, the I/O is completed by the device, and its driver

btt: Life of an I/O

- Q2I – time it takes to process an I/O *prior* to it being inserted or merged onto a request queue
  - Includes split, and remap time
- I2D – time the I/O is “idle” on the request queue
- D2C – time the I/O is “active” in the driver and on the device
- Q2I + I2D + D2C = Q2C
  - Q2C: Total processing time of the I/O

btt: Partial Output

DEV	#Q	#D	Ratio	BLKmin	BLKavg	BLKmax	Total
[8, 0]	92827	12401	7.5	1	109	1024	10120441
[8, 1]	93390	13676	6.8	1	108	1024	10150343
[8, 2]	92366	13052	7.1	1	109	1024	10119302
[8, 3]	92278	13616	6.8	1	109	1024	10119043
[8, 4]	92651	13736	6.7	1	109	1024	10119903

DEV	Q2I	I2D	D2C	Q2C
[8, 0]	0.049697430	0.302734498	0.074038617	0.400079555
[8, 1]	0.031665593	0.050032148	0.058669682	0.125934697
[8, 2]	0.035651772	0.031035436	0.047311659	0.096735504
[8, 3]	0.021047776	0.011161007	0.038519804	0.060975408
[8, 4]	0.028985217	0.008397228	0.034344640	0.058160497

DEV	Q2I	I2D	D2C
[8, 0]	11.7%	71.0%	17.4%
[8, 1]	22.6%	35.6%	41.8%
[8, 2]	31.3%	27.2%	41.5%
[8, 3]	29.8%	15.8%	54.5%
[8, 4]	40.4%	11.7%	47.9%

Merge Ratio:
#Q / #D

"Software" time

Avg I/O time

SCSI bus, target:
low- to high-priority

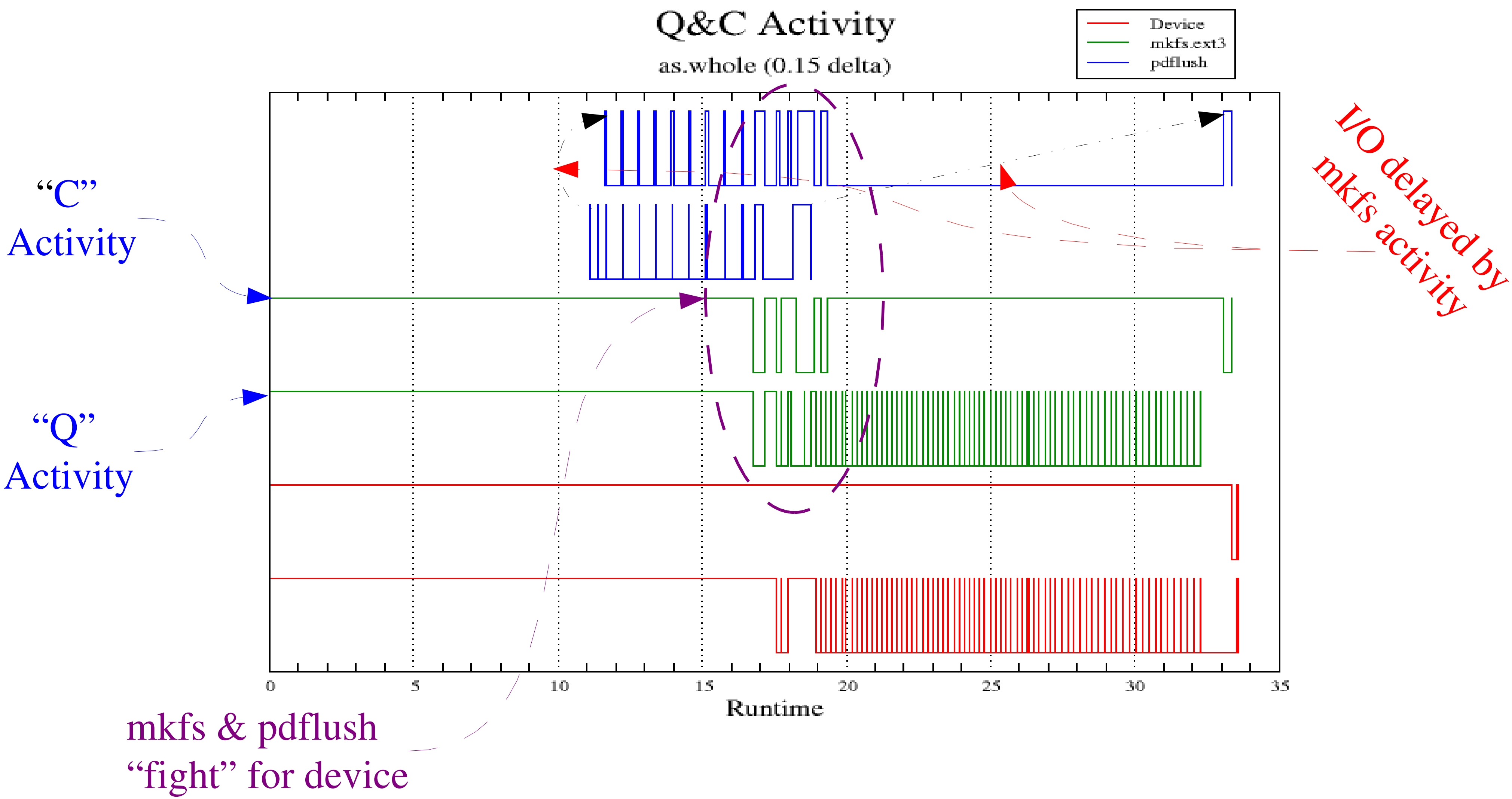
Driver/Device time

Excessive "idle" time on
request queue

btt: Q&C Activity

- *btt* also generates "activity" data – indicating ranges where processes and devices are actively handling various events (block I/O entered, I/O inserted/merged, I/O issued, I/O complete, ...)
- This data can be plotted (e.g. xmgrace) to see patterns and extract information concerning anomalous behavior

btt: I/O Scheduler Example



btt: I/O Scheduler - Explained

- Characterizing I/O stack
- Noticed very long I2D times for certain processes
- Graph shows continuous stream of I/Os...
  - ...at the device level
  - ...for the *mkfs.ext3* process
- Graph shows significant lag for *pdflush* daemon
  - Last I/O enters block I/O layer around 19 seconds
  - But: last batch of I/Os aren't completed until 14 seconds later!
- Cause? Anticipatory scheduler – allows mkfs.ext3 to proceed, holding off pdflush I/Os

Resources

- Kernel sources:
  - Patch for Linux 2.6.14-rc3 (or later, up to 2.6.17)
  - Linux 2.6.17 (or later) – built in
- Utilities & documentation (& kernel patches)
  - rsync://rsync.kernel.org/pub/scm/linux/kernel/git/axboe/blktrace.git
  - See documentation in *doc* directory
- Mailing list: linux-btrace@vger.kernel.org