Linux Kernel IO subsystem

How it works and how can I see what is it doing?

Jan Kára

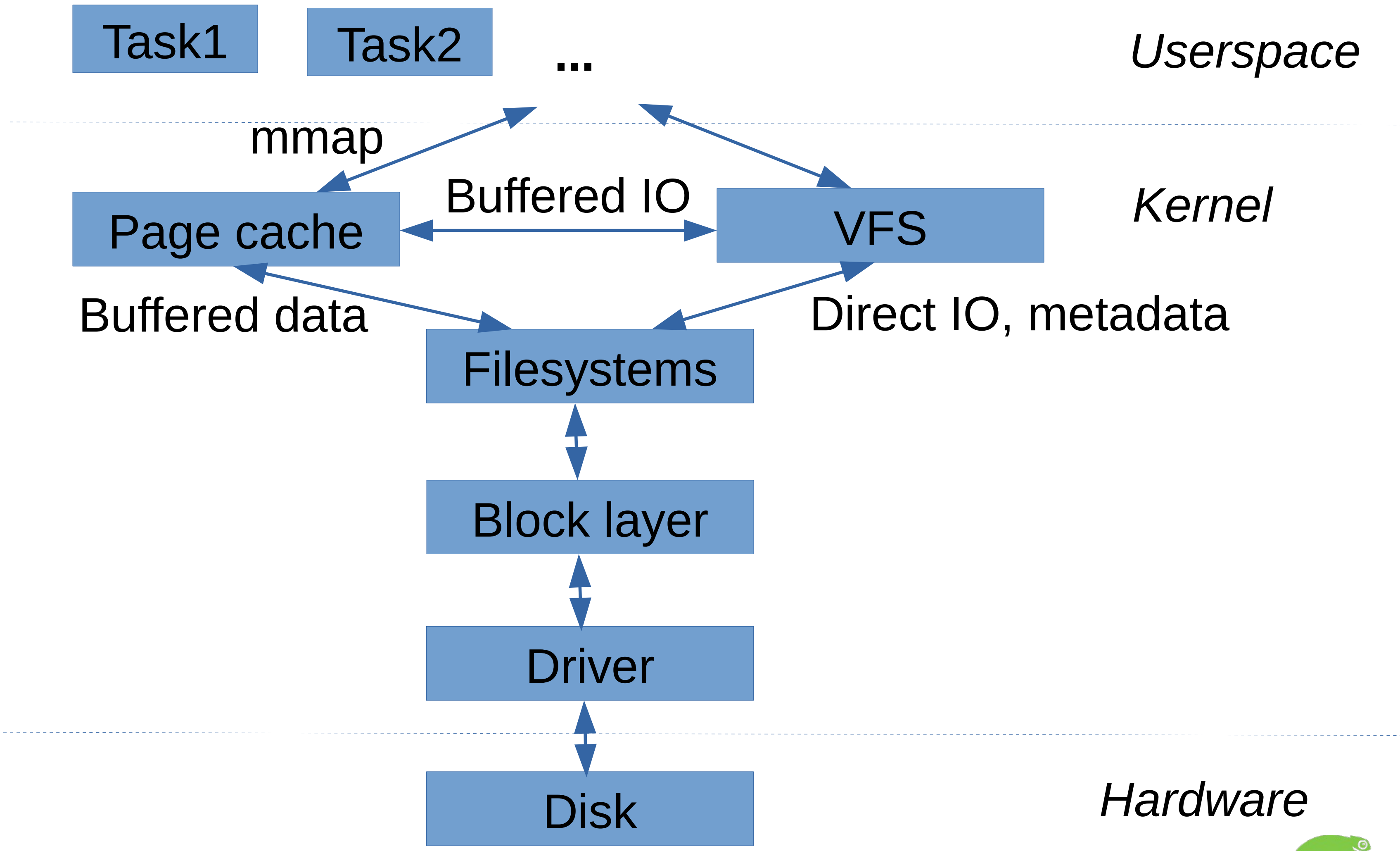
jack@suse.cz

Outline

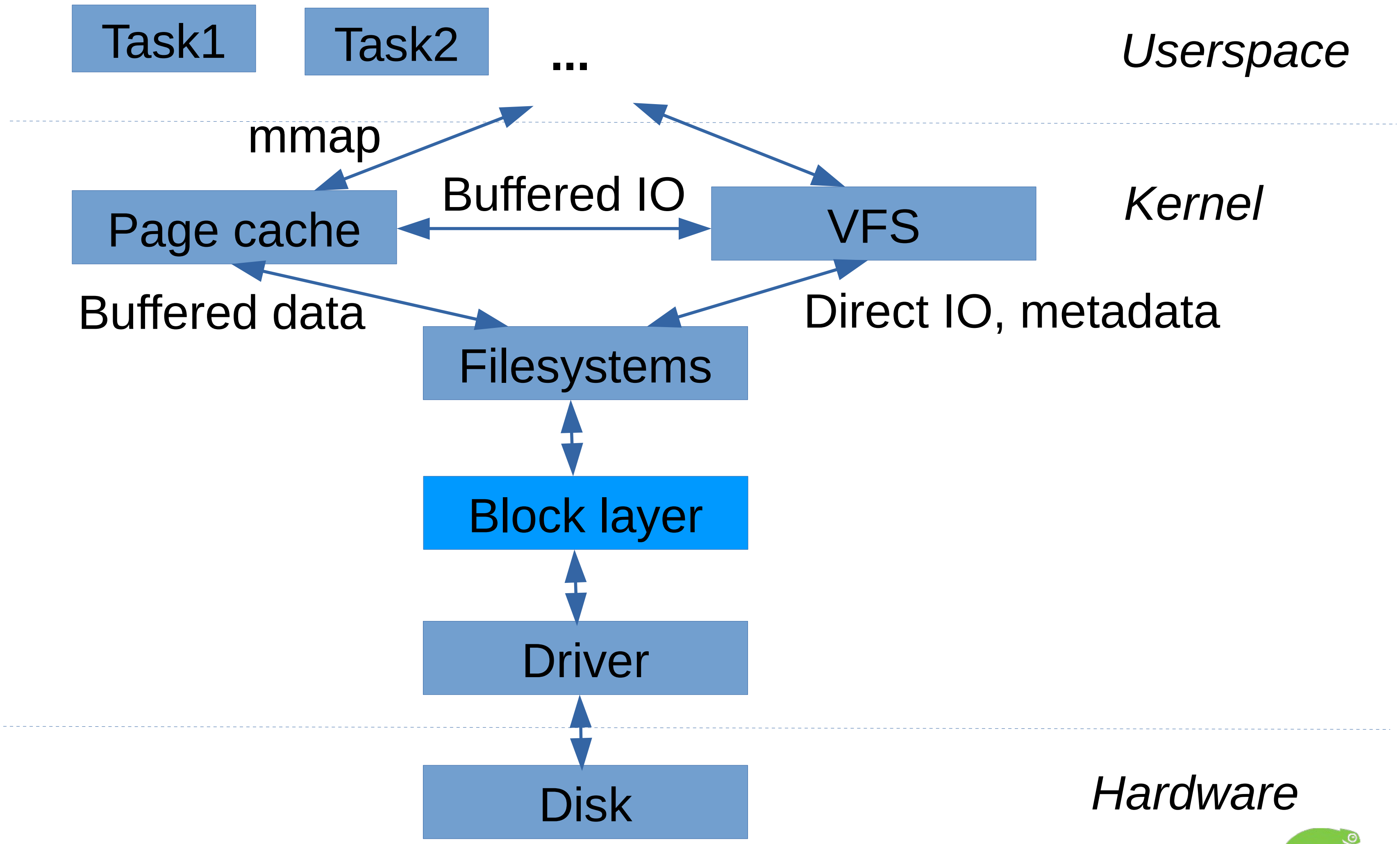
- Basic architecture of Linux IO subsystem
- Tools
- Examples

Linux Kernel IO Subsystem

Linux Kernel IO Architecture



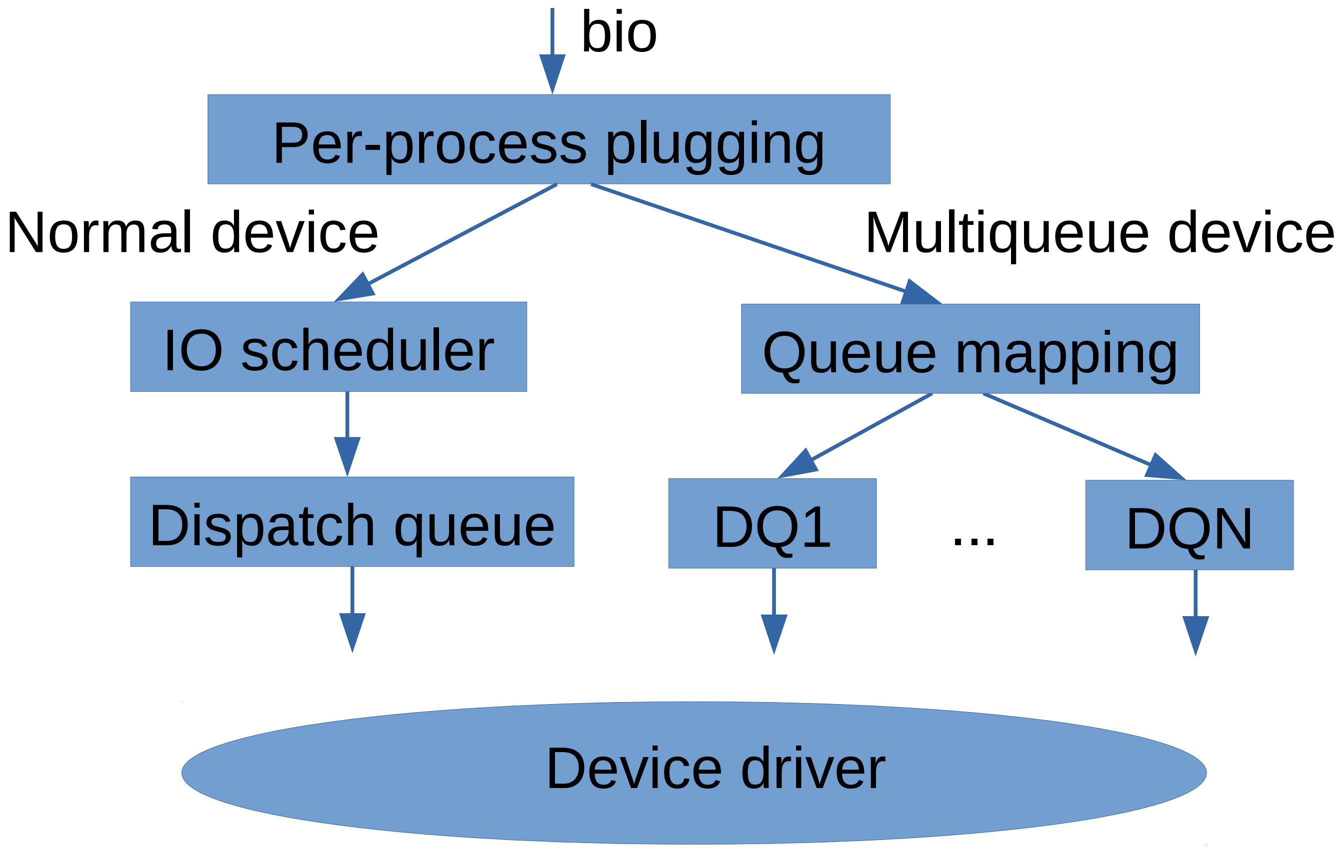
Linux Kernel IO Architecture



Block Layer Basics

- Works with IO requests
  - Starting sector, length, read / write / special
  - Can have hints (SYNC) and other flags (FUA, FLUSH)
- Life of a request
  - Created in block layer when IO submitted by a filesystem
  - Can be delayed, merged (IO scheduler, multiqueue handling)
  - Dispatched into a device driver
  - Completed when IO is finished

Submission Handling in Block Layer



IO Schedulers

- Decide when and in which order IO requests are submitted
  - NOOP – just pass requests into dispatch queue
  - Deadline
    - Prefers reads over writes
    - Sorts waiting requests to reduce seeking
    - Aims to dispatch each request at latest after its deadline has expired
  - CFQ
    - Prefers sync requests over async
    - Tries to achieve fairness among tasks
    - Support for IO priorities, cgroups, sync request idling, ...

Multiqueue Device Handling

- Used for fast devices
- Limited plugging
- No IO scheduling
- Support for multiple hardware IO queues
- Lightweight

IO Performance Analysis

Iostat

usually packaged in sysstat package

- Shows basic statistics about IO
- Very lightweight
- Run: iostat -dxk 1

```
Dev:   rrqm/s wrqm/s    r/s   w/s   rkB/s  wkB/s avgrq-sz avgqu-sz await svctm
sda    182.00   0.00 353.00  0.00 2152.00   0.00    12.19     0.89  2.53  2.50
dm-0     0.00   0.00 536.00  0.00 2144.00   0.00     8.00     1.42  2.65  1.74

Dev:   rrqm/s wrqm/s    r/s   w/s   rkB/s  wkB/s avgrq-sz avgqu-sz await svctm
sda    235.00   0.00 300.00  0.00 2160.00   0.00    14.40     0.87  2.89  2.86
dm-0     0.00   0.00 540.00  0.00 2160.00   0.00     8.00     2.01  3.72  1.79
```

Example: Storage attached via Xen

- Sequential writes slow when writing through Xen blkfront driver to multipathed device
- Direct: 112 MB/s, throught Xen: 46 MB/s
- Iostat numbers:

Dev:	rrqm/s	wrqm/s	r/s	w/s	rkB/s	wkB/s	avgrq-sz	avgqu-sz	await	svctm
Sdb	0.00	0.00	0.00	354.00	0.00	176128.00	995.07	31.97	91.79	2.84

IO through Xen

Dev:	rrqm/s	wrqm/s	r/s	w/s	rkB/s	wkB/s	avgrq-sz	avgqu-sz	await	svctm
Sdd	0.00	0.00	0.00	1377.00	0.00	59988.00	87.13	30.98	22.38	0.73

After plugging fix (104 MB/s):

Dev:	rrqm/s	wrqm/s	r/s	w/s	rkB/s	wkB/s	avgrq-sz	avgqu-sz	await	svctm
Sdb	0.00	0.00	0.00	571.00	0.00	145920.00	511.10	31.66	55.68	1.76

Takeaway 1

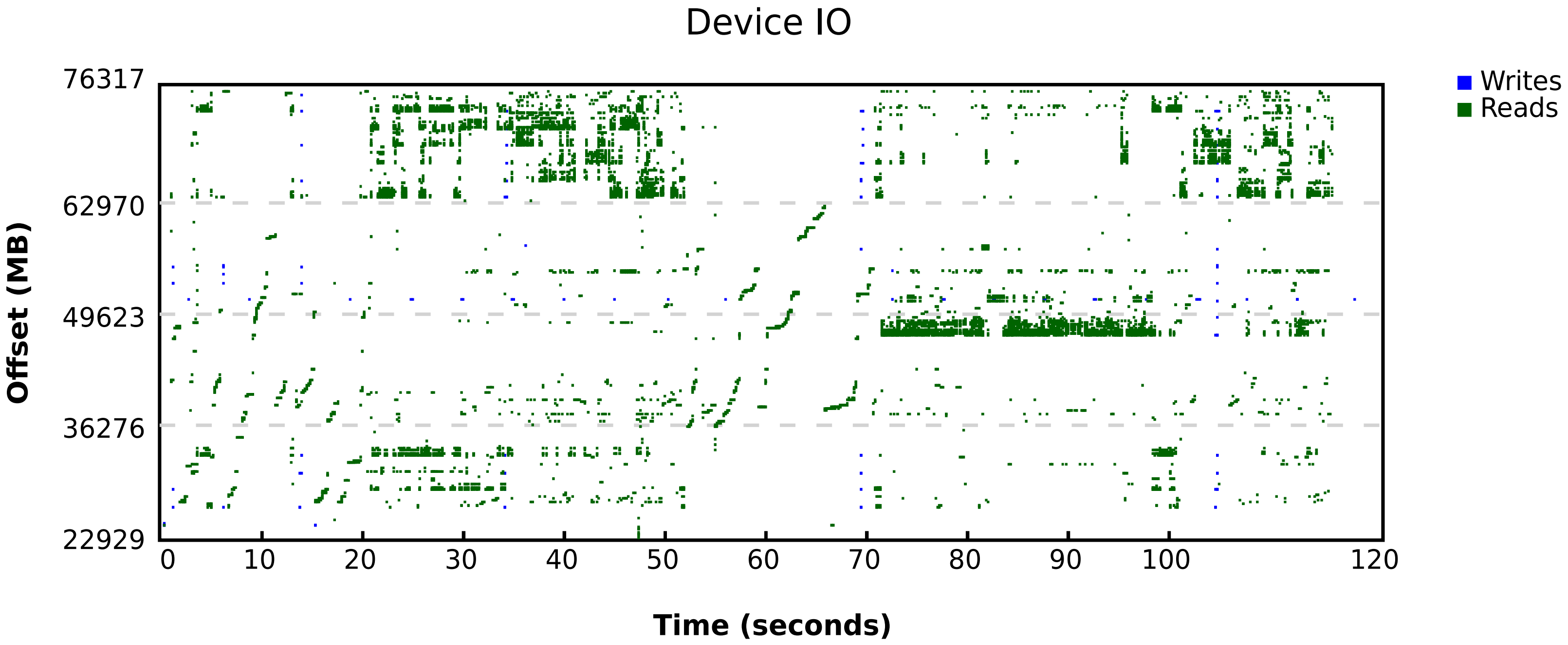
- Small requests hurt throughput
  - Overhead in kernel
  - Overhead in the device itself
- Holds for any storage device

Iowatcher

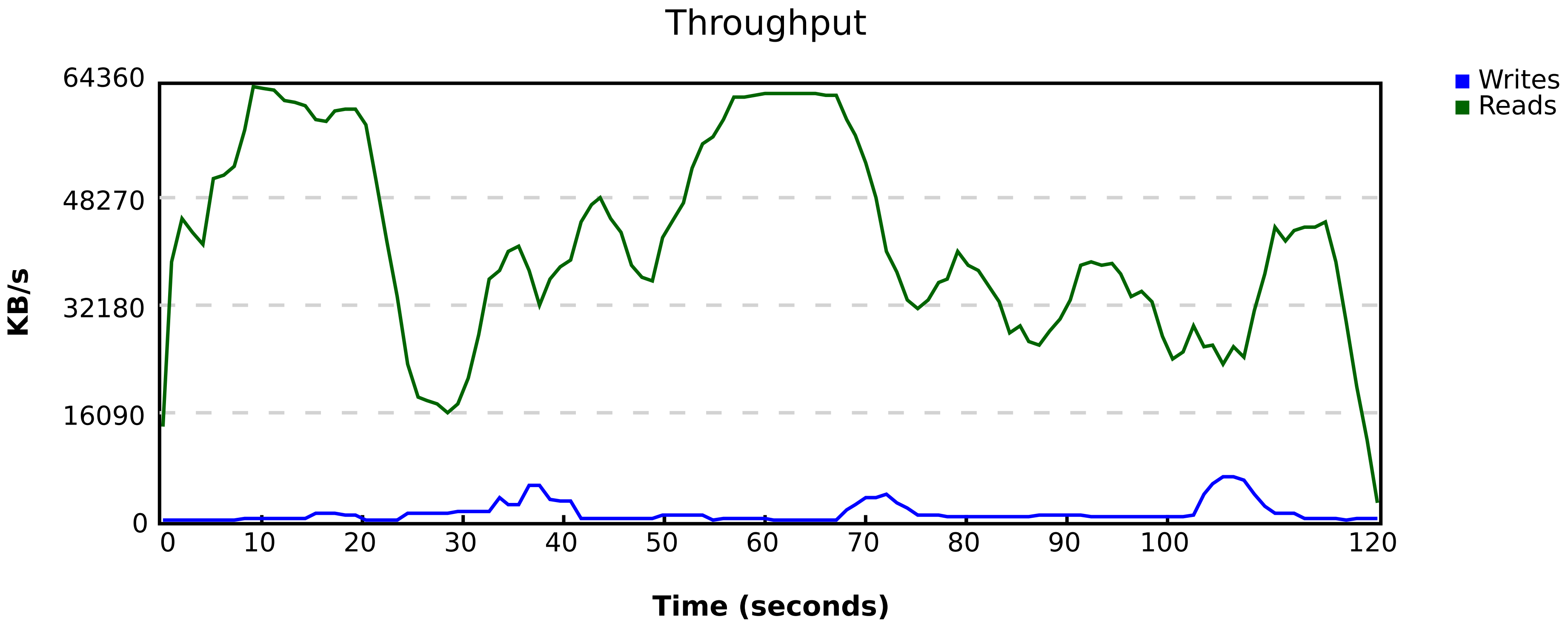
http://masoncoding.com/iowatcher/

- Consumes traces from blktrace, mpstat output, fio bandwidth logs
- Plots various graphs (throughput, IO latency, IO location, …)
- Movies

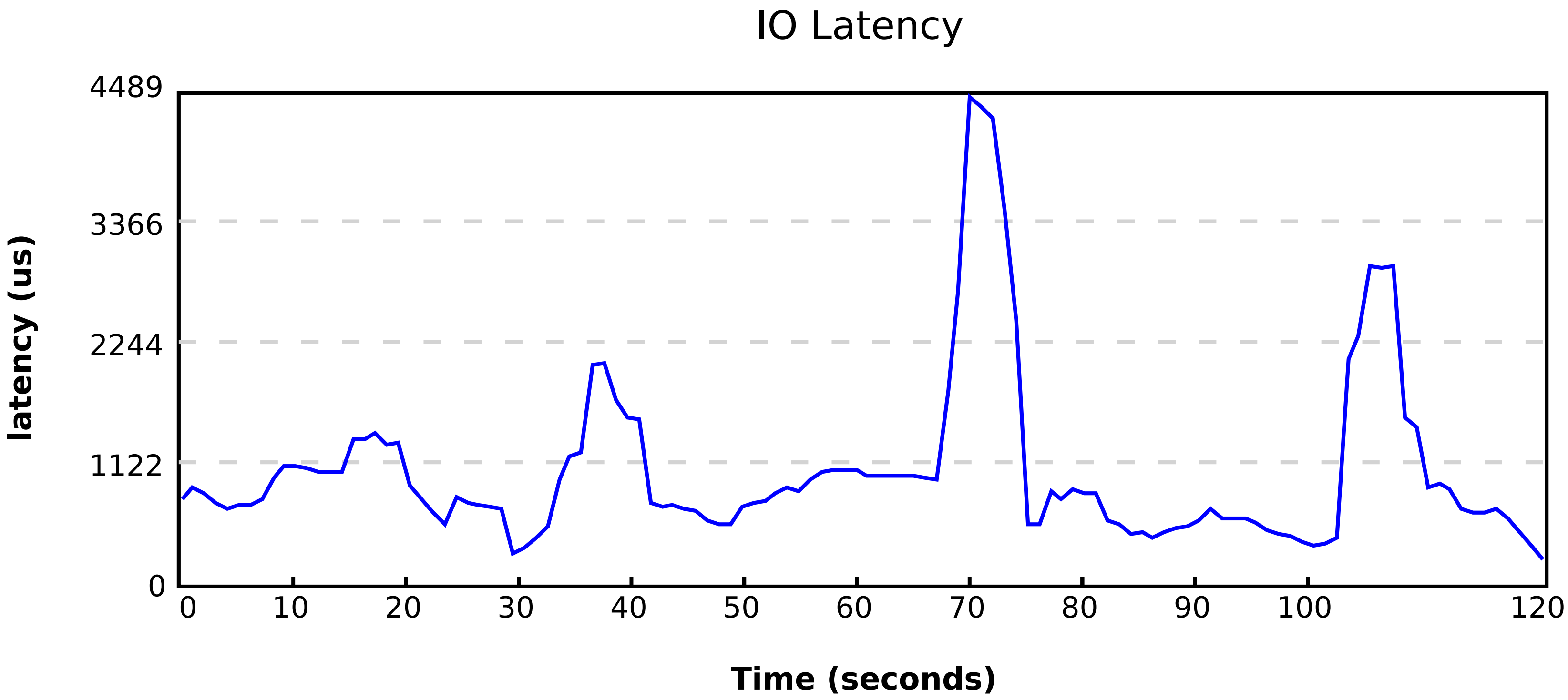
Iowatcher Graphs



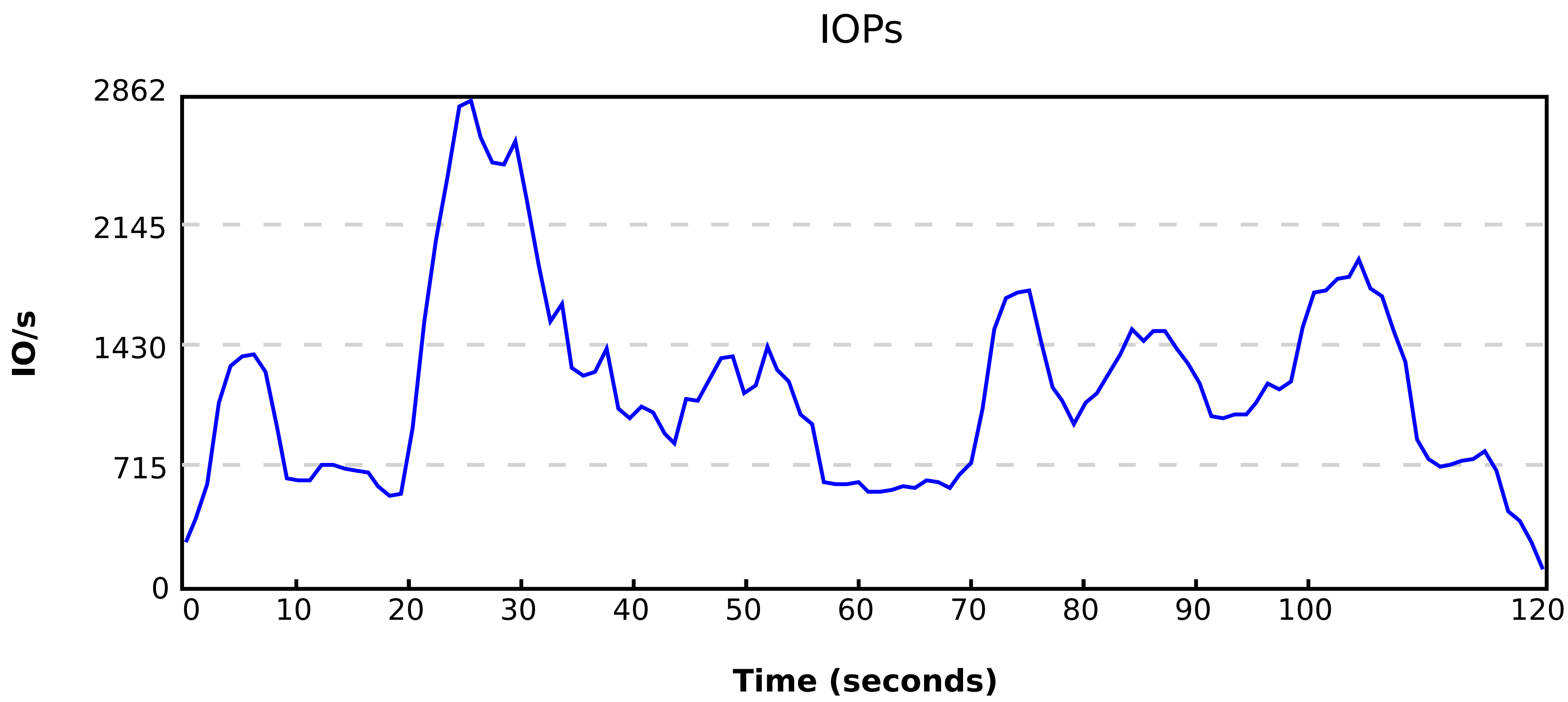
Iowatcher Graphs



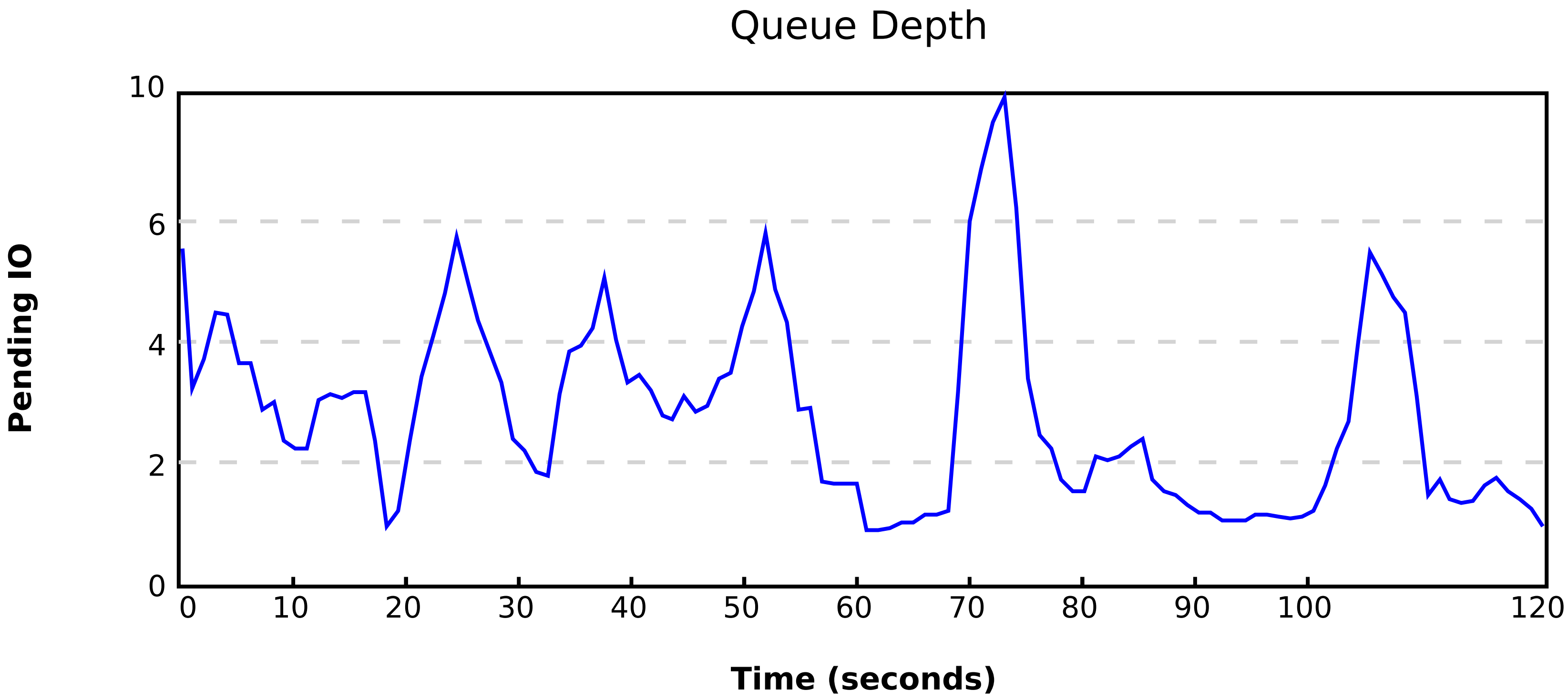
Iowatcher Graphs



Iowatcher Graphs



Iowatcher Graphs



Iowatcher Movie

- Movie time!

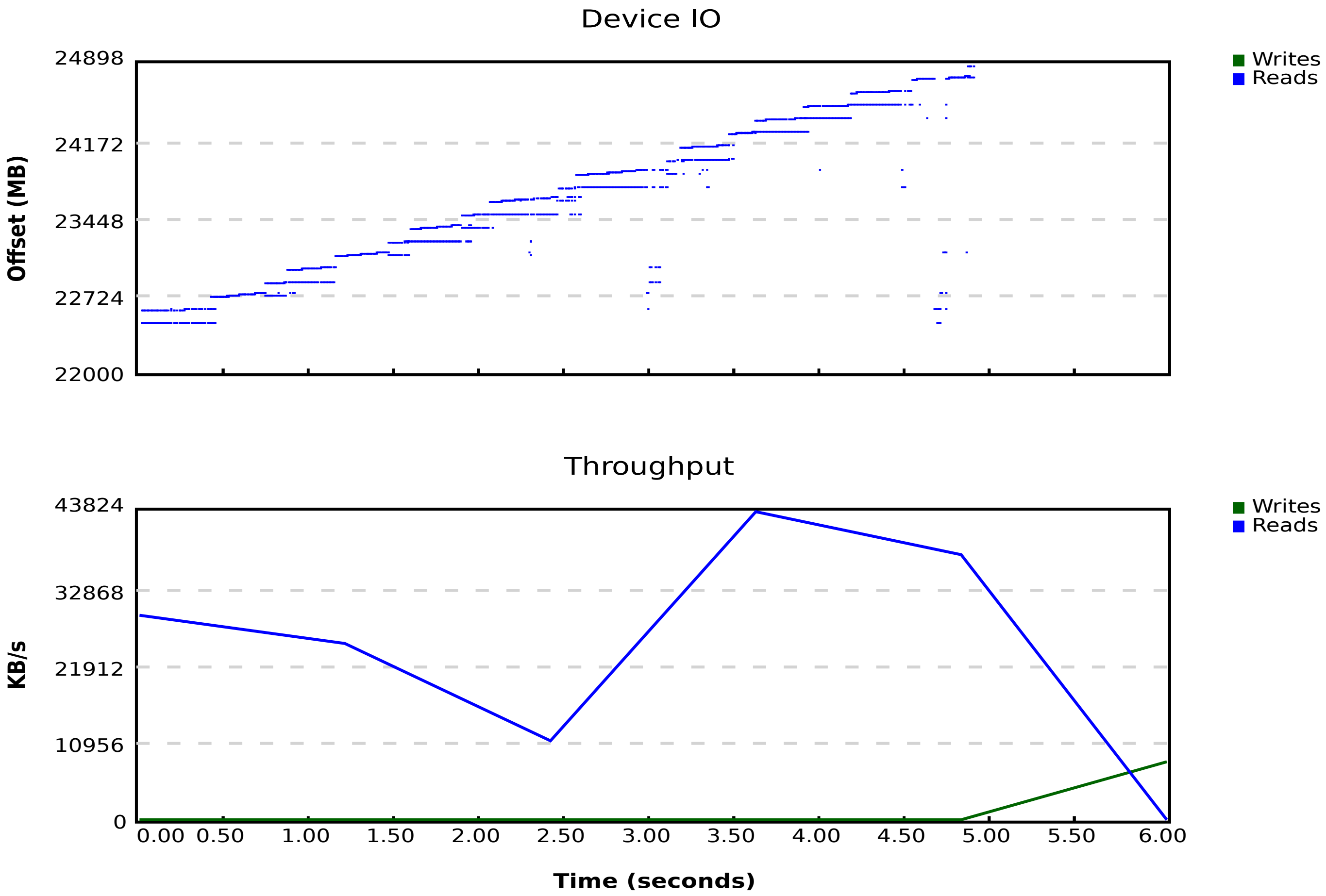
Iowatcher Use Example

- Enabling ext4 dir_index feature slows down deleting of a directory tree.

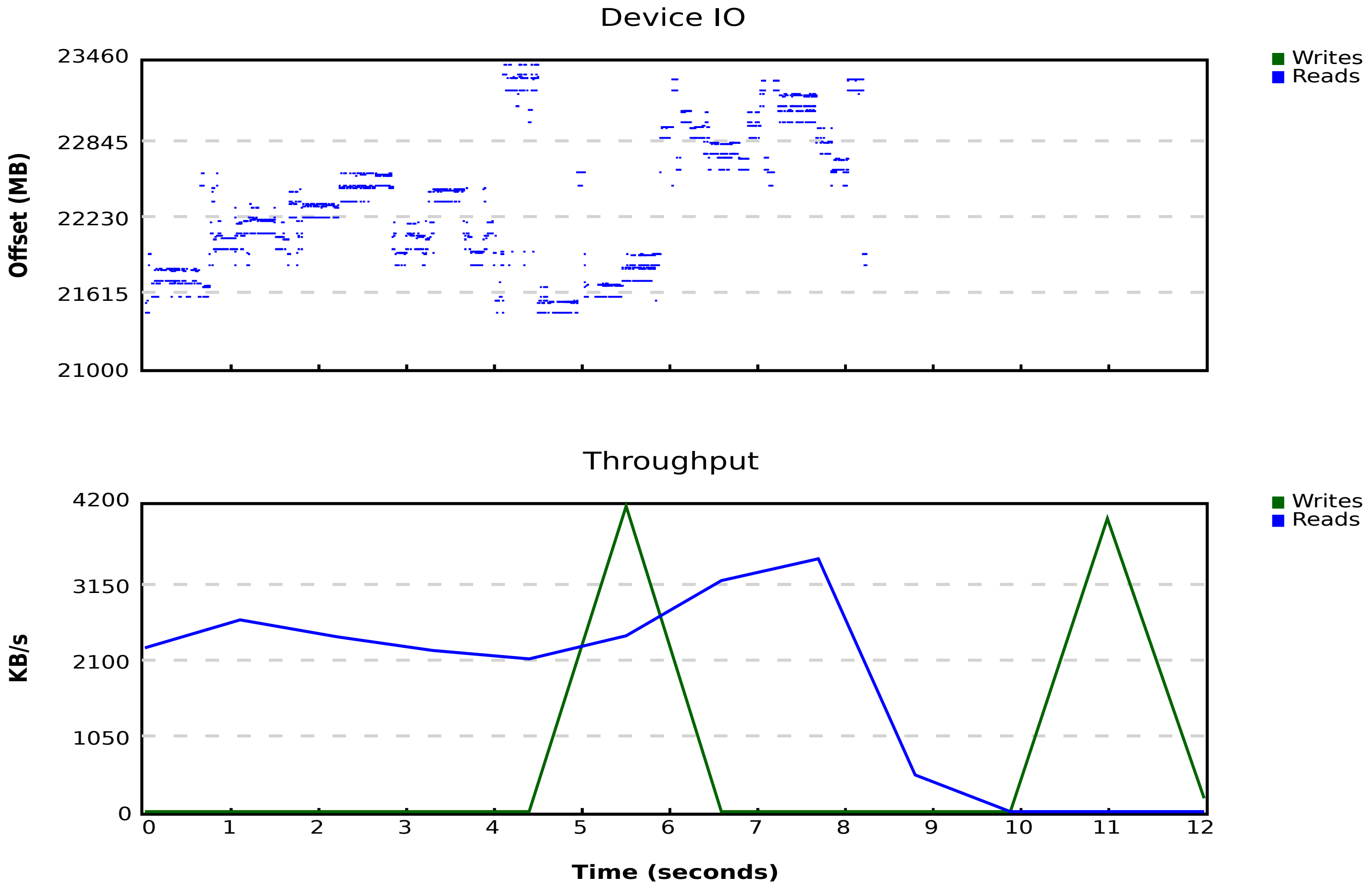
```
# Without dir_index
leet:~ # time rm -r /mnt/linux-2.6.32.orig/
real  0m4.862s
user  0m0.032s
sys   0m2.388s

# With dir_index
leet:~ # time rm -r /mnt/linux-2.6.32.orig/
real  0m8.100s
user  0m0.040s
sys   0m2.588s
```

Tree Deletion without Dir_index



Tree Deletion with Dir_index



Takeaway 2

- Seeking matters
  - Smaller chances of merging IO requests
  - Seek time for rotational storage

Blktrace

- Detailed information about IO requests processing
- Relatively large amount of data
  - May store over network
- Can handle multiple devices in parallel
- Gather as:

```
blktrace -d /dev/sda -d /dev/sdb -d /dev/dm-0
```

- Further processing using iowatcher, blkparse, btt

Blkparse

```
8,0 4 498  0.536245624 5072  A RM 46664392 + 8 <- (8,1) 46662344
8,0 4 499  0.536248072 5072  Q RM 46664392 + 8 [gcc]
8,0 4   0  0.536262021    0  m  N cfq5072S / alloced
8,0 4 500  0.536262739 5072  G RM 46664392 + 8 [gcc]
8,0 4 501  0.536266614 5072  I RM 46664392 + 8 [gcc]
8,0 4   0  0.536268520    0  m  N cfq5072S / insert_request
8,0 4   0  0.536270374    0  m  N cfq5072S / add_to_rr
8,0 4   0  0.536276200    0  m  N cfq workload slice:75
8,0 4   0  0.536278314    0  m  N cfq5072S / set_active wl_prio:0
                                    wl_type:2
8,0 4   0  0.536280939    0  m  N cfq5072S / fifo=(null)
8,0 4   0  0.536282276    0  m  N cfq5072S / dispatch_insert
8,0 4   0  0.536285224    0  m  N cfq5072S / dispatched a request
8,0 4   0  0.536286509    0  m  N cfq5072S / activate rq, drv=1
8,0 4 502  0.536286919 5072  D RM 46664392 + 8 [gcc]
8,0 4 503  0.556455119    0  C RM 46664392 + 8 [0]
8,0 4   0  0.556469202    0  m  N cfq5072S / complete rqnoidle 0
8,0 4   0  0.556471881    0  m  N cfq5072S / set_slice=25
8,0 4   0  0.556475942    0  m  N cfq5072S / arm_idle: 2
                                    group_idle: 0
8,0 4   0  0.556476510    0  m  N cfq schedule dispatch
```

Blkparse (cont)

```
8,0 0    1  0.556659272 5073  A  R 47002176 + 32 <- (8,1) 47000128
8,0 0    2  0.556661415 5073  Q  R 47002176 + 32 [gcc]
8,0 0    0  0.556674617    0  m  N cfq5073S / alloced
8,0 0    3  0.556675354 5073  G  R 47002176 + 32 [gcc]
8,0 0    4  0.556677319 5073  P  N [gcc]
8,0 0    5  0.556680380 5073  I  R 47002176 + 32 [gcc]
8,0 0    0  0.556682649    0  m  N cfq5073S / insert_request
8,0 0    0  0.556684273    0  m  N cfq5073S / add_to_rr
8,0 0    6  0.556688402 5073  U  N [gcc] 1
8,0 4    0  0.564839523    0  m  N cfq idle timer fired
8,0 4    0  0.564842003    0  m  N cfq5072S / slice expired t=0
8,0 4    0  0.564844568    0  m  N / served: vt=9817282560
                                     min_vt=9817278464
8,0 4    0  0.564847483    0  m  N cfq5072S / sl_used=2 disp=1
                                     charge=2 iops=0 sect=8
8,0 4    0  0.564849177    0  m  N cfq5072S / del_from_rr
8,0 4    0  0.564850534    0  m  N cfq schedule dispatch
8,0 4    0  0.564869775    0  m  N cfq5073S / set_active wl_prio:0
                                     wl_type:2
8,0 4    0  0.564871692    0  m  N cfq5073S / fifo=(null)
8,0 4    0  0.564872827    0  m  N cfq5073S / dispatch_insert
8,0 4    0  0.564875317    0  m  N cfq5073S / dispatched a request
8,0 4    0  0.564876421    0  m  N cfq5073S / activate rq, drv=1
8,0 4  504  0.564876894 2743  D  R 47002176 + 32 [kworker/4:2]
8,0 2   86  0.570193124    0  C  R 47002176 + 32 [0]
```

Btt

- Uses binary event dump of blktrace events
- Produces various statistics
  - Q2C, Q2D, D2C latencies
  - Current device queue depth
  - Seeks per second
  - Per process activity
- Useful to check what to look for in blkparse output

Ftrace

- General kernel tracing framework
- Controlled via /sys/kernel/debug/tracing
- `Documentation/trace/ftrace.txt`
- Static trace points
  - Syscalls
  - Various events in filesystems, writeback, …
- Dynamic trace points on almost every kernel function

Complex Problem Analysis

Problem

- When customer launches a large tarball creation, HA monitors of postgress DB occasionally time out ⇒ service failover
  - Service timeout 10 seconds
- Used HW raid for storage, deadline IO scheduler, ext3 filesystem
- 8 GB of memory free, disk is loaded with writes

Analysis 1/7

- Iostat pretty normal:

```
Dev rrqm/s   wrqm/s  r/s    w/s rkB/s    wkB/s avgrq-sz avgqu-sz
Sda   0.00 13726.00 1.00 174.00  2.00 43876.00   501.46   142.12
```

- Blktrace output large (~900 MB parsed)
- Use btt to show latency outliers
  – Watch out for lost blktrace event

```
sort -k 2 -n -r q2clat.dat | head -30
127.842616 10.341348
127.842619 10.341346
127.842621 10.341344
```

…

Analysis 2/7

- Match back to blktrace events using timestamps

```
8,0 118.779433534 638 A WS 11705240 + 8 <- (8,2) 11395984
8,0 118.779433858 638 Q WS 11705240 + 8 [kjournald]
8,0 118.779435324 638 G WS 11705240 + 8 [kjournald]
8,0 118.779436253 638 I WS 11705240 + 8 [kjournald]
…
8,0 123.784506489   0 D WS 11705240 + 8 [swapper]
…
8,0 125.870800714   0 C WS 11705240 + 8 [0]
```

- Ok, so some IOs really take long
- Deadline IO scheduler delays outlaying IO

Analysis 3/7

- Switched IO scheduler to NOOP
- Max latency reduced 3.8s and generally better
- Service time outs increased !?!
- Need more insight into why they time out
- Use ftrace to trace syscalls

```
echo 1 >/sys/kernel/debug/tracing/events/syscalls/enable
cat /.../tracing/trace_pipe | gzip -c >syscall-trace.gz
```

Analysis 4/7

- Found large latencies in fsync

```
postgres-17461 [001]  3559.059091: sys_fsync(fd: 4)
postgres-17461 [008]  3570.848573: sys_fsync -> 0x0
syslog-ng-3008 [005]  3433.451593: sys_fsync(fd: 7)
syslog-ng-3008 [005]  3449.854534: sys_fsync -> 0x0
```

…

- Partly caused by heavy logging from sysrq-w
- Partly design limitation of data=ordered mode of ext3

Analysis 5/7

- Removed sysrq-w, switched ext3 to data=writeback
- Time outs less frequent but still occur
- Another syscall trace analysis

```
crm_master-20388 [000] 355206.448764: sys_read(...)
...
crm_node-20389 [006] 355207.654087: sys_mmap(...)
crm_node-20389 [006] 355207.654091: sys_mmap -> ...
crm_node-20389 [006] 355208.889691: sys_close(fd: 3)
crm_node-20389 [006] 355208.889693: sys_close -> 0x0
...
crm_master-20388 [001] 355220.880237: sys_read -> 0xa
```

Analysis 6/7

- Correlate inactivity periods with blktrace

```
8,0 283.784399307  1867  A   R 9373112 + 8 <- (8,2) 9063856
8,0 283.784399608  1867  Q   R 9373112 + 8 [crm_node]
8,0 283.784400643  1867  G   R 9373112 + 8 [crm_node]
8,0 283.784401175  1867  P   N [crm_node]
8,0 283.784401701  1867  I   R 9373112 + 8 [crm_node]
8,0 283.784402232  1867  U   N [crm_node] 1
…
8,0 284.987422579     0  D   R 9373112 + 8 [swapper]
…
8,0 284.995404698     0  C   R 9373112 + 8 [0]
```

Analysis 7/7

- Slow reads of shared libs sum up to time out
- Reads behind writes
- Switch back to deadline IO scheduler, tune deadlines and fifo_batch much lower
- Finally services run reliably

Conclusion

Conclusion

- Complex interactions between storage, block layer, filesystems
- Watch out for unexpected small requests, seeks
- Dependent reads vs async writes
  - Hard to guarantee latency under load

Thank You

We adapt. You succeed.