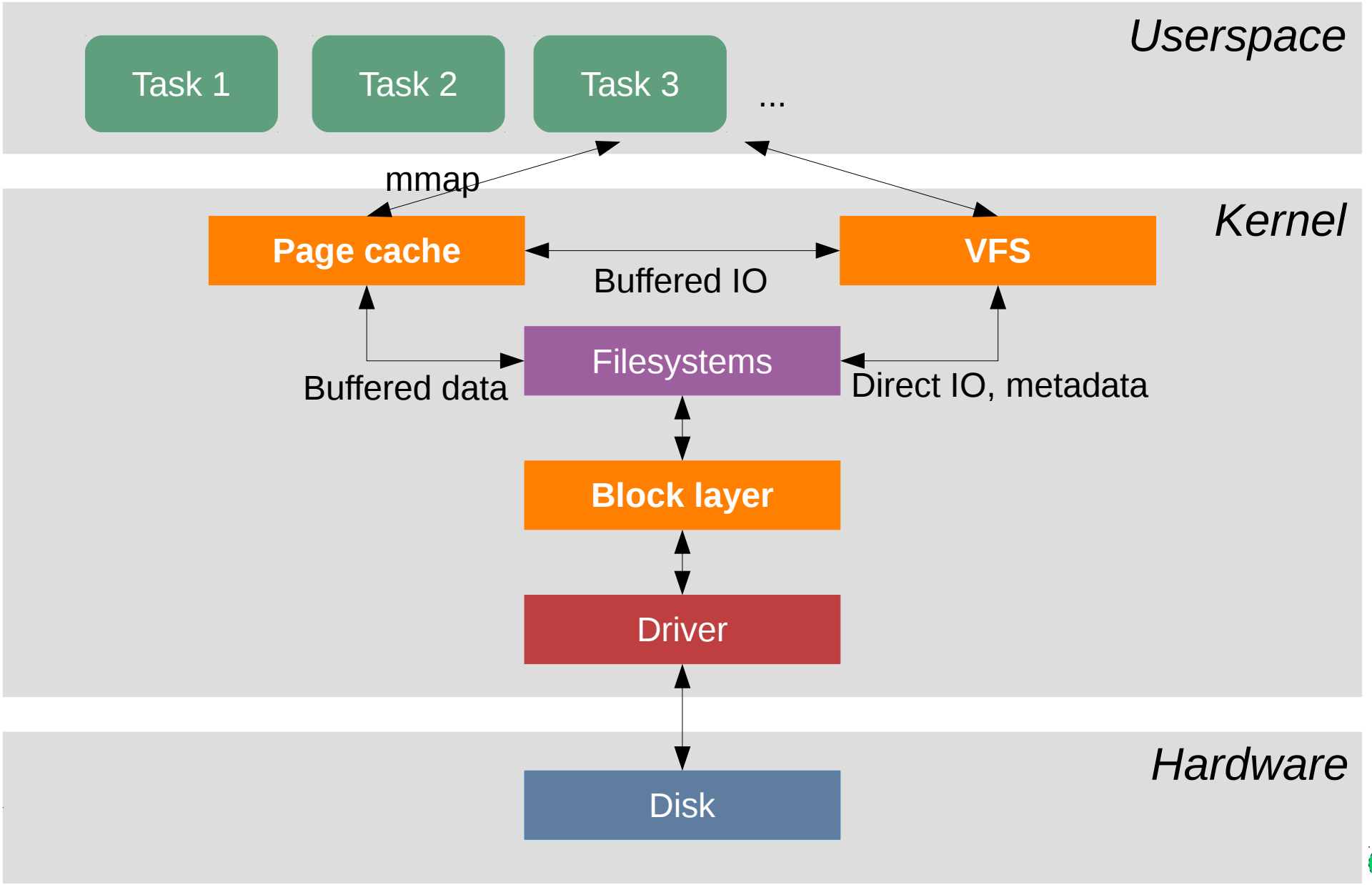
Case Study – IO Performance

Beijing Trace Training 2017

Gary Lin
Software Engineer, SUSE Labs
glin@suse.com

IO Subsystem



Virtual File System (VFS)

- Permission Check
- File name resolution
- Caching of directory hierarchy (*dentry cache*)
- Caching of inodes (*inode cache*)
- Management of file descriptors

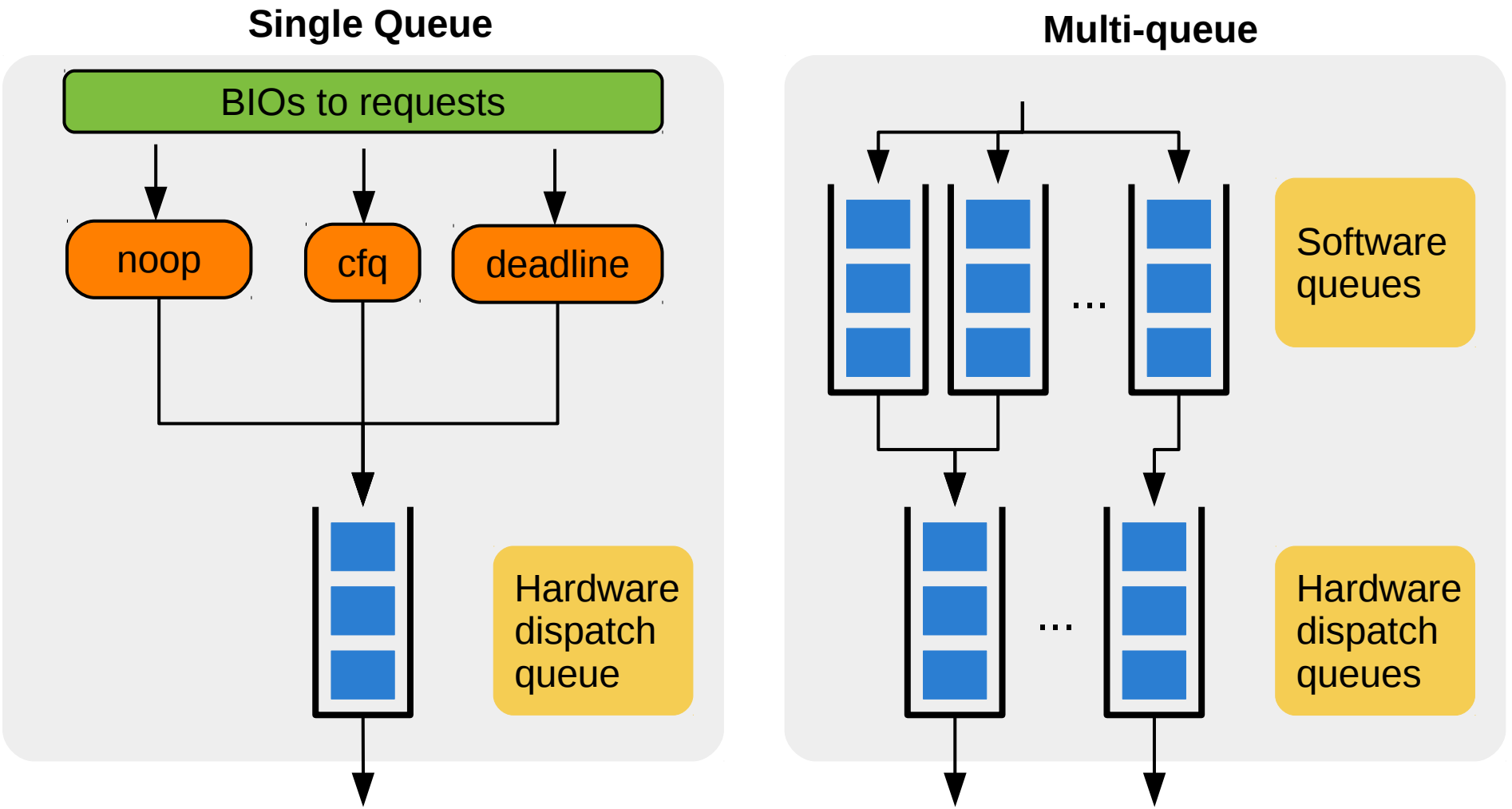
Page Cache

- Caching file data for standard read and write operations
- Reading file data in advance (*readahead*)
- Writing new data from page cache (*writeback*)

Block Layer

- Handling IO requests from file systems (IO scheduling)
  - Mapping IO requests to the target devices and location
  - Merging or splitting incoming IO requests
  - Delaying some requests to improve locality of IO
  - Giving some IO requests priority over others
- Two modes of operation of the block layer: **single queue** mode and **multi-queue** mode

Block Layer (cont'd)



IO Performance Tools

iostat

- Gather basic statistics about issued IO
- Very lightweight and suitable for long term monitoring
- The statistics of disks are available in
  /proc/diskstats and /sys/block/<dev>/stat
- Example:

```
$ iostat -x

Device: rrqm/s wrqm/s  r/s  w/s rkB/s  wkB/s avgrq-sz avgqu-sz
sda       0.01   0.52 1.40 2.69 85.72 117.65    99.55     0.44
         await r_await w_await  svctm  %util
        107.70   15.58  155.63   6.10   2.49
```

Display the statistics every 10 seconds

```
$ iostat -x 10
```

iostat fields

`rrqm/s:` read requests merged per second
`wrqm/s:` write requests merged per second

`r/s:` read requests completed per second
`w/s:` write requests completed per second

`rkB/s:` the number of kilobytes read from the device per second
`wkB/s:` the number of kilobytes written to the device per second

`avgrq-sz:` the average size (in sectors) of the requests
`avgqu-sz:` the average queue length of requests

iostat fields (cont’d)

`await:` the average time (in ms) for I/O requests to be served

`r_await:` the average time (in ms) for read requests to be served

`w_await:` the average time (in ms) for write requests to be served

`svctm:` the average service time (in ms) for I/O requests that were issued to the device

`%util:` Percentage of elapsed time during which I/O requests were issued to the device

blktrace

- A tool to gather detailed information about request queue operations up to user space
- blktrace enables the tracing mechanism with the ioctl code BLKTRACESETUP and gather the binary output from `/sys/kernel/debug/block/<dev>/trace<cpu>`.
- To reduce the influence of tracing, it's good to store resulting traces on a *separate* disk or send the result over network to a *different* host.
- The bundled viewers: *blkparse*, *iowatcher*, and *btt*
- Example:

```
# blktrace -d /dev/sda -o <path to other disk>
```

blkparse

- A utility to parse the events from *blktrace* and convert them to human-readable form
- Example:

  Parse the trace in the post-processing mode

```
# blkparse -i sda.blktrace.* -o sda-trace.txt
```

  Trace the events lively

```
# blktrace -d /dev/sda -o - | blkparse -i -
```

blkparse – Output

```
Dev CPU Seq        Time  PID Act Dir Sector + len
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 – Output

```
Dev CPU Seq        Time  PID Act Dir Sector + len
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
```

remap from sda1 to sda

Add to queue

Allocate request

Send to IO scheduler

Send to driver

Done

blkparse – Actions

A – a request is remapped

Q – a request will be added to the queue

G – a request structure is being allocated

I – a request was passed to the IO scheduler

D – a request was passed to the device driver

C – a request was completed

blkparse – Request Direction

- Request Direction
  - **R** – read
  - **W** – write
  - **D** – discard
  - **N** – none
- Additional Modifier
  - **S** – synchronous
  - **M** – metadata
  - **F** – cache flush

iowatcher

- Create visualizations from blktrace results
- Example

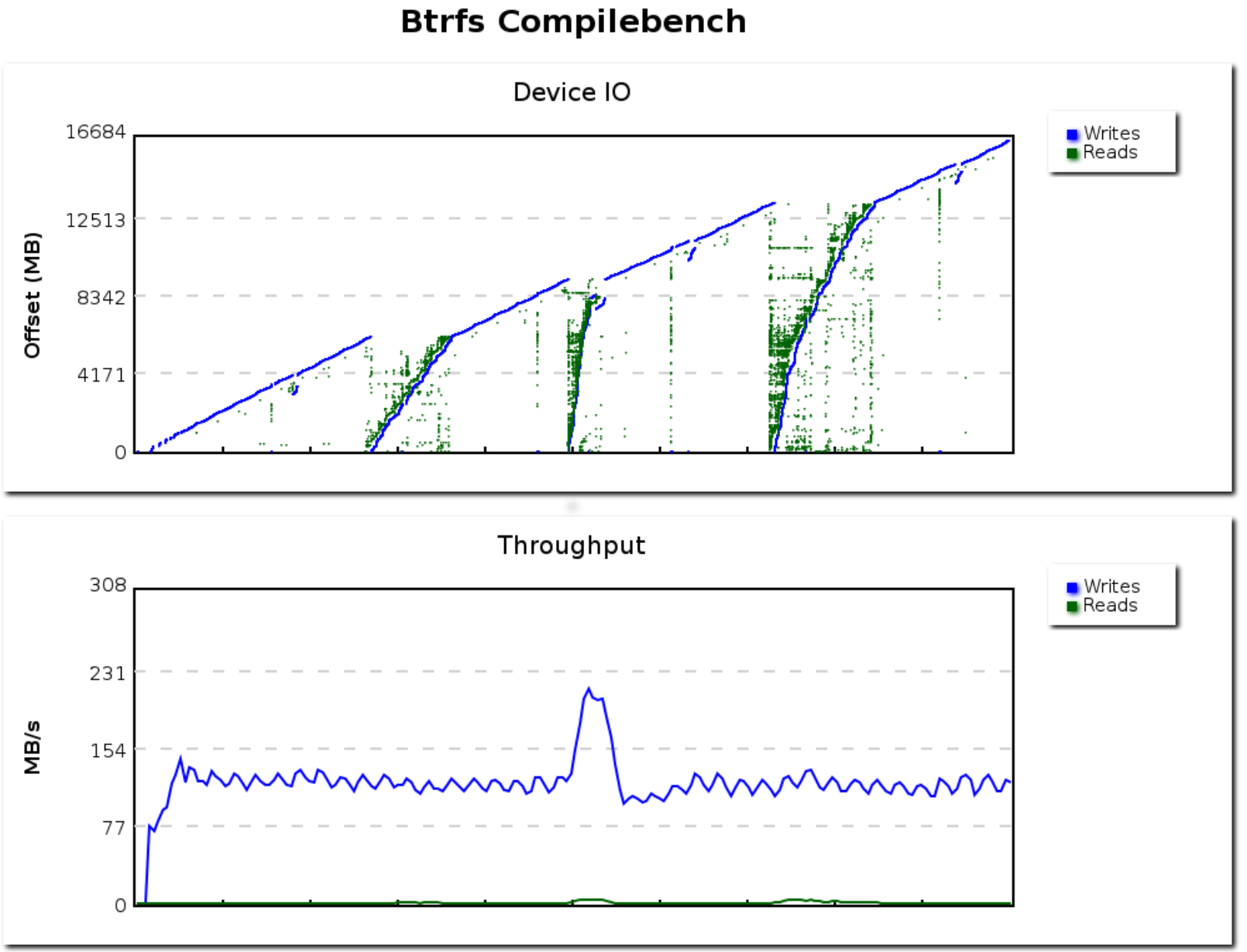
  Draw a static graph

```
$ iowatcher -t sda.blktrace.0 -o sda.svg
```

  Output a video

```
$ iowatcher -t sda.blktrace.0 --movie -o sda.mp4
```

iowatcher – Output



btt

- Generate various statistics from blktrace data
- Example:

```
$ btt -i sda.blktrace.* -o sda.btt
$ ls
  8,0_iops_fp.dat 8,0_mbps_fp.dat sda.btt.avg
  sda.btt.dat sda.btt_dhist.dat sda.btt.msg
  sda.btt_qhist.dat sys_iops_fp.dat sys_mbps_fp.dat
```

btt – Output

```
==================== All Devices ====================

            ALL            MIN            AVG            MAX            N
-------------- ------------ ------------ ------------ -----------

Q2Q               0.000003352   0.022244663  11.054857755         1858
Q2G               0.000001048   0.000003126   0.000091495         1631
G2I               0.000000838   0.000506987   0.009771682         1841
Q2M               0.000000838   0.000001661   0.000028915          228
I2D               0.000001955   0.000744232   0.156676199         1681
M2D               0.000022699   0.000468931   0.012235957          266
D2C               0.000146391   0.012406427   0.127382126          838
Q2C               0.000162944   0.051680212  34.306738827          901
```

btt – Latencies

- **Q2G**

  time from a block I/O is queued to the time it gets a request allocated for it

- **G2I**

  time from a request is allocated to the time it is inserted into the device queue

- **I2D**

  time from a request is inserted into the device queue to the time it is actually issued to the device

- **D2C**

  service time of the request by the device

- **Q2C**

  total time spent in the block layer for a request

Cases from Jan Kara

Case 1: Multipath Storage Attached Via Xen

“A customer reported an issue where a sequential write to a multipath storage achieved throughput of only **58MB/s** when access through **Xen guests** … **172MB/s** … from the **host** directly”

Analysis – iostat results

Host direct write:

```
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
```

Write via 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
```

Average time to complete IO per sector (`avgrq-sz/svctm`):

```
2.84/995.07 = 2.8 ms/sector (host)
0.73/87.13  = 8.4 ms/sector (Xen)
```

Smaller request size introduces higher management overhead.

Solution

- Delay the submission of IO inside the Xen driver
- Result:

  Throughput of Xen guest: 142MB/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
```

Case 2: IOzone Write Regression

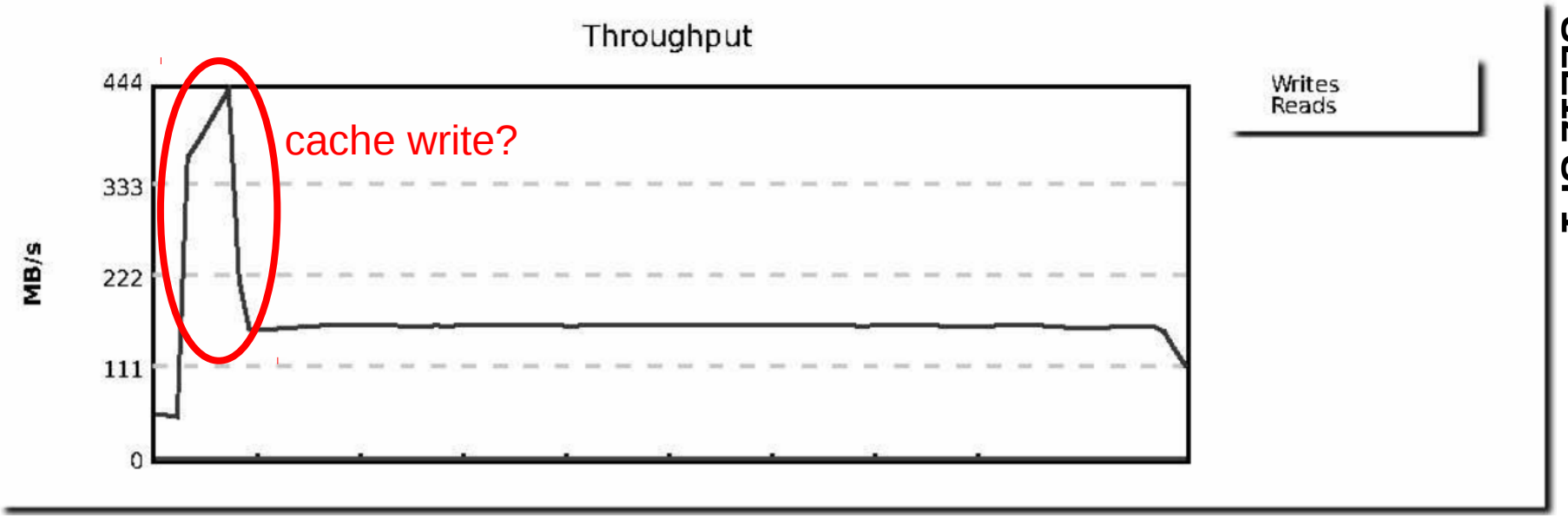
“Our QA has reported … iozone benchmark has reported sequential writes are 10-25% slower for 8GB file in SLE12 SP2 than they used to be in SLE12 SP1.”

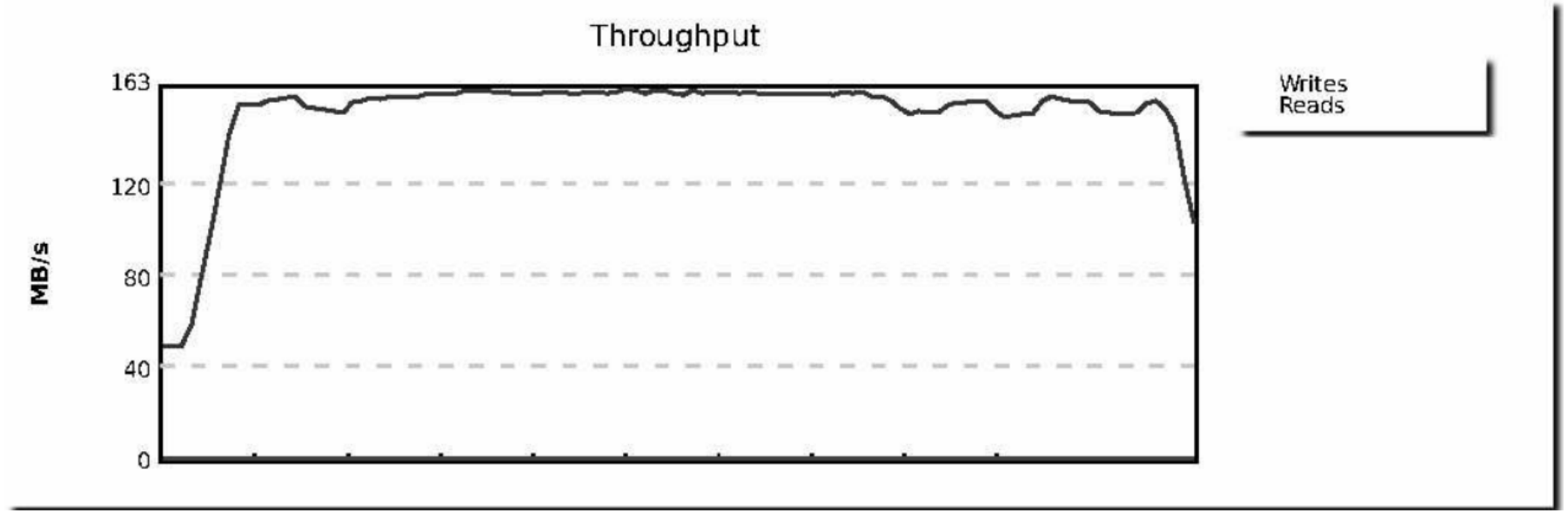
Analysis

- The regression was visible only on 2 machines out of 6.
- Suspicion:

  A caching problem in page cache or in the HW RAID card both machines had installed

Analysis – Throughput (iowatcher)





Analysis – blktrace

SLE12-SP1

```
Dev CPU Seq        Time  PID Act Dir Sector + len
8,0   8   5 0.000040600 3620   D   W 83900688 + 2048 [ext4lazyinit]
8,0   8  10 0.000120358 3620   D   W 83902736 + 2040 [ext4lazyinit]
8,0   8  17 0.001559611 3620   D   W 83904776 + 2048 [ext4lazyinit]
...
8,0   5 101 3.437315292 3493   D   W 84170752 + 2048 [kworker/u66:0]
8,0   5 102 3.437345162 3493   D   W 84172800 + 2048 [kworker/u66:0]
8,0   5 103 3.437369321 3493   D   W 84174848 + 2048 [kworker/u66:0]
8,0   5 104 3.437393603 3493   D   W 84176896 + 2048 [kworker/u66:0]
```

SLE12-SP2

```
8,0  29   5 0.000031108 3344   D   W 83925256 + 2048 [ext4lazyinit]
8,0  29  11 0.000755350    0   D  WS 83927304 + 2048 [swapper/0]
8,0  29  17 0.040032880 3344   D   W 83929352 + 2048 [ext4lazyinit]
...
8,0  17 681 3.730722333  516   D   W 84170752 + 8192 [kworker/u68:12]
8,0  17 695 3.730772903  516   D   W 84178944 + 8192 [kworker/u68:12]
8,0  17 696 3.730797989  516   D   W 84187136 + 8192 [kworker/u68:12]
8,0  17 697 3.730819719  516   D   W 84195328 + 8192 [kworker/u68:12]
```

Analysis – ext4lazyinit

- ext4 filesystem invokes `ext4lazyinit` for the delayed zeroing of inode tables.
  - These writes happen only shortly after the file system is created.
  - These writes are bad for benchmark since they are not the real workload.
- Both SLE12 SP1 and SLE12 SP2 had `ext4lazyinit` events, so they are not the root cause.

Analysis – Request Size

- By default, SLE12 SP1 uses 1MB request size while SLE12 SP2 uses 4MB request size.

  `/sys/block/<dev>/queue/max_sectors_kb`

- The regression disappeared after `max_sectors_kb` was reverted to 1MB.

Solution

- The HW RAID controller advertises optimal IO request size of 256KB.
- The patch was backported to match the HW optimal IO request size.

Case3: HA Monitor Timeouts

“A customer reported an issue where a high-availability monitor of postgress database occasionally times out when a **large tarball** is being created on the node.”

Analysis – Setup

- OS: SLE11-SP3
- Storage: Hardware RAID
- IO Scheduler: deadline
- Filesystem: ext3
- About 8GB of memory was free when the issue occurred.
- Iostat showed the disk was loaded with writes.
- The recorded blktrace is around 900MB.

Manual inspection is almost impossible!

Analysis – btt

- “`btt -q`” generated queue-to-completion latencies.
- Several IOs took several seconds to complete which were close to the timeout.

Analysis – blktrace

```
Dev            Time PID Act Dir Sector + len
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]
```

IO Scheduler
5 seconds

Device Write
2 seconds

All the write activities were stalled possibly leading to the stalls in the postgress database.

The situation was even worse after switching to **noop**.

Analysis – Syscalls

- Enable logging of all system calls using ftrace

```
# echo 1 > /sys/kernel/debug/tracing/events/syscalls/enable
```

```
Process          CPU Time          Syscall
syslog-ng-3008 005 3433.451593: sys_fsync(fd: 7)
syslog-ng-3008 005 3449.854534: sys_fsync -> 0x0
...
postgres-17461 001 3559.059091: sys_fsync(fd: 4)
postgres-17461 008 3570.848573: sys_fsync -> 0x0
```

16 seconds

11 seconds

- The source of long latencies of `fsync` was partly due to the debugging logging into syslog-ng and ext3 data=ordered.

Solution – Tuning ext3

- By the design of ext3 data=ordered mode, fsync has to write all the data attached to the currently running transaction.
  - The transaction may contain a substantial amount of unrelated data due to merge.
- Removing debug logging and switching to data=**writeback** mode moderated the issue.

Analysis – Syscalls II

- Some inactive processes were found.

```
Process           CPU Time             Syscall
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
```

Reading from a pipe connected to crm_node: **14** seconds

1 second stall!

Analysis – blktrace II

```
Dev            Time  PID Act Dir Sector + len
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]
```

Queued for 1 second

Solution II – Tuning Deadline Scheduler

- The read stalls were caused by paging in of a shared library used by crm_node process.
- The read operations were queued due to the large amount of writes generated by the creation of the tarball.
- Tuning the deadline scheduler to avoid read requests from being starving:

```
/sys/block/<device>/queue/iosched/fifo_batch to 4 (16)
/sys/block/<device>/queue/iosched/read_expire to 200 (HZ/2)
/sys/block/<device>/queue/iosched/write_expire to 1000 (5*HZ)
/sys/block/<device>/queue/iosched/writes_starved to 4 (2)
                                                    HZ = 250
```

Question?


SUSE®
We adapt. You succeed.