



Untangling convoluted performance regression

Jan Kára <jack@suse.cz>
SUSE Labs



Introduction

Joint work with Anthony Iliopoulos from SUSE Labs

Example of complicated IO performance regression

Good demonstration of various tools one can use to debug IO regression

Also various snags and dead ends one can meet ... and overcome



XFS performance regression



The problem

IBM DB2 backup restore to XFS filesystem slowed down:

- SLE12-SP5 (4.12-based kernel, released 2019) - around 17 minutes
- SLE15-SP6 (6.4-based kernel, released 2024) - around 56 minutes

No problem when restoring to other filesystems (ext4, NFS)

No problem with simple tests writing file to the XFS filesystem

- In fact direct IO writes were 70% faster with SLE15-SP6 (270 vs 463 MB/s)

Challenges

Very big difference between baseline and test kernel

- 4.12 vs 6.4 (about 5 years of difference)

Proprietary load

- Unclear how the load actually behaves
- Unable to reproduce locally

Virtualized in VMware so unclear performance characteristics

Initial investigation

Different feature sets between SLE12-SP5 and SLE15-SP6 filesystems

```
meta-data=/dev/mapper/vg_datvg1-lv_db2ERXdata1 isize=512    agcount=4, agsize=72089344 blks
          =                               sectsz=512    attr=2, projid32bit=1
          =                               crc=1          finobt=0 spinodes=0 rmapbt=0
          =                               reflink=0
data      =                               bsize=4096    blocks=288357376, imaxpct=5
          =                               sunit=0        swidth=0 blks
naming    =version 2                      bsize=4096    ascii-ci=0 ftype=1
log        =internal                      bsize=4096    blocks=140799, version=2
          =                               sectsz=512    sunit=0 blks, lazy-count=1
realtime  =none                           extsz=4096    blocks=0, rtextents=0
```

```
meta-data=/dev/mapper/vg_datvg1-lv_db2ERXdata1 isize=512    agcount=4, agsize=72089344 blks
          =                               sectsz=512    attr=2, projid32bit=1
          =                               crc=1          finobt=1, sparse=1, rmapbt=1
          =                               reflink=1       bigtime=1 inobtcount=0 nnext64=0
data      =                               bsize=4096    blocks=288357376, imaxpct=5
          =                               sunit=0        swidth=0 blks
naming    =version 2                      bsize=4096    ascii-ci=0, ftype=1
log        =internal log                  bsize=4096    blocks=140799, version=2
          =                               sectsz=512    sunit=0 blks, lazy-count=1
realtime  =none                           extsz=4096    blocks=0, rtextents=0
```

Initial investigation

Suspicion for writeback throttling (blk-wbt)

- Framework to throttle background writeback if read latency crosses configured threshold
- Enabled by default in SLE15-SP6 (and later) for multiqueue devices
- Tracked several regressions in write intensive loads to that

Writeback throttling was indeed enabled in the system but disabling it didn't help



Tracing

Gathered iostat(1) output and stacktraces of blocked tasks (sysrq-w)

- Confirmed read side is not an issue
- Throughput degrades from ~160 MB/s to ~30 MB/s as the restore runs in SLE15-SP6
- SLE12-SP5 drives much higher queue depth (19 vs 4)
- Direct IO used to restore the DB - puzzling

SLE12-SP5

Device	wkB/s	aqu-sz	wareq-sz
sdd	218856	11	31.37
sdd	243763	57	31.24
sdd	220191	52	31.21
sdd	242201	51	31.26

SLE15-SP6

Device	wkB/s	aqu-sz	wareq-sz
sdd	139334	1	28.99
sdd	156424	16	29.30
...			
sdd	39972	3	20.02

Tracing

Gathered blktrace(8) to get detailed insight into IO pattern

Big difference in amount of metadata IO between SLE12-SP5 and SLE15-SP6

- 7% vs 30% of IO

IO heavily parallel with ~170 tasks submitting writes

Much larger delays between IO completion and submission of next IO in SLE15-SP6 (12ms vs 30ms)



Where does the metadata difference come from?

XFS filesystem statistics

SLE12-SP5

extent_alloc 3795 74761029 2624 30369

blk_map 7017436 7016859 56 7016797 56 14034413 0

trans 2 13382318 0

log 426083 23443547 0 7005398 13584729

push_ail 13382320 0 20625 2064 0 805 2 430 0 34

rw 7016842 181

buf 12668399 76608 12665090 377 146486 3309 0 76548 13

abtb2 6477 11164 856 727 0 0 0 0 0 0 0 0 0 0 3414

abtc2 11463 19322 5796 5667 0 0 0 0 0 0 0 0 0 0 14714

fi2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

rmapbt 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

SLE15-SP6

extent_alloc 4411 74761645 4315 32060

blk_map 7017328 7017053 56 7016859 56 14034561 0

trans 2 19686623 0

log 2899197 97973807 0 6621023 10597950

push_ail 19686631 0 59155 12583 0 2428 102 413 0 174

rw 7016842 181

buf 58166624 7707 58158917 42796493 150138 7707 0 ...

abtb2 11234 20730 2003 1875 0 0 0 0 0 0 0 0 0 0 12567

abtc2 22786 44252 9465 9337 0 0 0 0 0 0 0 0 0 0 38799

fi2 386 382 4 4 0 0 0 0 0 0 0 0 0 0 0

rmapbt 16071736 222035901 4061827 3864718 34 26 ...

Where does the metadata difference come from?

XFS filesystem statistics

SLE12-SP5

extent_alloc 3795 74761029 2624 30369

blk_map 7017436 7016859 56 7016797 56 14034413 0

trans 2 13382318 0

log 426083 23443547 0 7005398 13584729

push_ail 13382320 0 20625 2064 0 805 2 430 0 34

rw 7016842 181

buf 12668399 76608 12665090 377 146486 3309 0 76548 13

abtb2 6477 11164 856 727 0 0 0 0 0 0 0 0 0 0 0 0 3414

abtc2 11463 19322 5796 5667 0 0 0 0 0 0 0 0 0 0 14714

fib2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

rmapbt 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

SLE15-SP6

extent_alloc 4411 74761645 4315 32060

blk_map 7017328 7017053 56 7016859 56 14034561 0

trans 2 19686623 0

log 2899197 97973807 0 6621023 10597950

push_ail 19686631 0 59155 12583 0 2428 102 413 0 174

rw 7016842 181

buf 58166624 7707 58158917 42796493 150138 7707 0 ...

abtb2 11234 20730 2003 1875 0 0 0 0 0 0 0 0 0 0 0 0 12567

abtc2 22786 44252 9465 9337 0 0 0 0 0 0 0 0 0 0 0 0 38799

fib2 386 382 4 4 0 0 0 0 0 0 0 0 0 0 0 0

rmapbt 16071736 222035901 4061827 3864718 34 26 ...

The same amount of writes and block mappings, similar amount of allocations

Where does the metadata difference come from?

XFS filesystem statistics

SLE12-SP5

extent_alloc 3795 74761029 2624 30369

blk_map 7017436 7016859 56 7016797 56 14034413 0

trans 2 13382318 0

log 426083 23443547 0 7005398 13584729

push_ail 13382320 0 20625 2064 0 805 2 430 0 34

rw 7016842 181

buf 12668399 76608 12665090 377 146486 3309 0 76548 13

abtb2 6477 11164 856 727 0 0 0 0 0 0 0 0 0 0 3414

abtc2 11463 19322 5796 5667 0 0 0 0 0 0 0 0 0 0 14714

fib2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

rmapbt 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

SLE15-SP6

extent_alloc 4411 74761645 4315 32060

blk_map 7017328 7017053 56 7016859 56 14034561 0

trans 2 19686623 0

log 2899197 97973807 0 6621023 10597950

push_ail 19686631 0 59155 12583 0 2428 102 413 0 174

rw 7016842 181

buf 58166624 7707 58158917 42796493 150138 7707 0 ...

abtb2 11234 20730 2003 1875 0 0 0 0 0 0 0 0 0 0 12567

abtc2 22786 44252 9465 9337 0 0 0 0 0 0 0 0 0 0 38799

fib2 386 382 4 4 0 0 0 0 0 0 0 0 0 0 0

rmapbt 16071736 222035901 4061827 3864718 34 26 ...

Significantly increased number of transactions and logged blocks

Where does the metadata difference come from?

XFS filesystem statistics

SLE12-SP5

extent_alloc 3795 74761029 2624 30369

blk_map 7017436 7016859 56 7016797 56 14034413 0

trans 2 13382318 0

log 426083 23443547 0 7005398 13584729

push_ail 13382320 0 20625 2064 0 805 2 430 0 34

rw 7016842 181

buf 12668399 76608 12665090 377 146486 3309 0 76548 13

abtb2 6477 11164 856 727 0 0 0 0 0 0 0 0 0 0 3414

abtc2 11463 19322 5796 5667 0 0 0 0 0 0 0 0 0 0 14714

fib2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

rmapbt 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

SLE15-SP6

extent_alloc 4411 74761645 4315 32060

blk_map 7017328 7017053 56 7016859 56 14034561 0

trans 2 19686623 0

log 2899197 97973807 0 6621023 10597950

push_ail 19686631 0 59155 12583 0 2428 102 413 0 174

rw 7016842 181

buf 58166624 7707 58158917 42796493 150138 7707 0 ...

abtb2 11234 20730 2003 1875 0 0 0 0 0 0 0 0 0 0 12567

abtc2 22786 44252 9465 9337 0 0 0 0 0 0 0 0 0 0 38799

fib2 386 382 4 4 0 0 0 0 0 0 0 0 0 0 0

rmapbt 16071736 222035901 4061827 3864718 34 26 ...

Big increase in number of buffer lookups, locks,...

Where does the metadata difference come from?

XFS filesystem statistics

SLE12-SP5

extent_alloc 3795 74761029 2624 30369

blk_map 7017436 7016859 56 7016797 56 14034413 0

trans 2 13382318 0

log 426083 23443547 0 7005398 13584729

push_ail 13382320 0 20625 2064 0 805 2 430 0 34

rw 7016842 181

buf 12668399 76608 12665090 377 146486 3309 0 76548 13

abtb2 6477 11164 856 727 0 0 0 0 0 0 0 0 0 0 3414

abtc2 11463 19322 5796 5667 0 0 0 0 0 0 0 0 0 0 14714

fib2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

rmapbt 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

SLE15-SP6

extent_alloc 4411 74761645 4315 32060

blk_map 7017328 7017053 56 7016859 56 14034561 0

trans 2 19686623 0

log 2899197 97973807 0 6621023 10597950

push_ail 19686631 0 59155 12583 0 2428 102 413 0 174

rw 7016842 181

buf 58166624 7707 58158917 42796493 150138 7707 0 ...

abtb2 11234 20730 2003 1875 0 0 0 0 0 0 0 0 0 0 12567

abtc2 22786 44252 9465 9337 0 0 0 0 0 0 0 0 0 0 38799

fib2 386 382 4 4 0 0 0 0 0 0 0 0 0 0 0

rmapbt 16071736 222035901 4061827 3864718 34 26 ...

Reverse map btree enabled in SLE15-SP6

Intermediate summary

Application seems to behave the same between SLE12-SP5 and SLE15-SP6 ✓

Much more (~4x as much) log IO in SLE15-SP6 ✗

Much lower IO parallelism in SLE15-SP6 (~5x difference in avg. in flight requests) ✗

Other differences in XFS stats with unclear origin ?

There were many code changes between the kernels but none clearly matching the observations ?

We are now some 3.5 months into the investigation...

Attempts at reproduction

Tried to replicate customer setup and load

- KVM VM with SSD disk with comparable throughput
- FIO random write load with appropriate block size (from blktrace) from multiple processes to multiple files

Managed to reproduce ~2x throughput difference between SLE12-SP5 and SLE15-SP6

- Also reproduces higher amount of logging load in SLE15-SP6
- Big part attributed to reverse map btree feature in SLE15-SP6 - win?

Attempts at reproduction

Tried to replicate customer setup and load

- KVM VM with SSD disk with comparable throughput
- FIO random write load with appropriate block size (from blktrace) from multiple processes to multiple files

Managed to reproduce ~2x throughput difference between SLE12-SP5 and SLE15-SP6

- Also reproduces higher amount of logging load in SLE15-SP6
- Big part attributed to reverse map btree feature in SLE15-SP6 - win?
 - Not quite - forgot the customer tested with rmapbt disabled and it didn't change a thing...

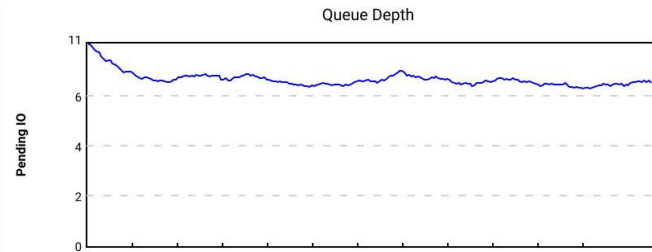
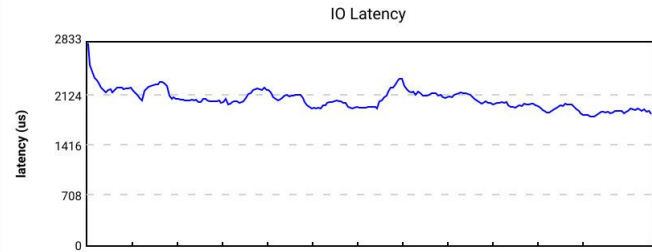
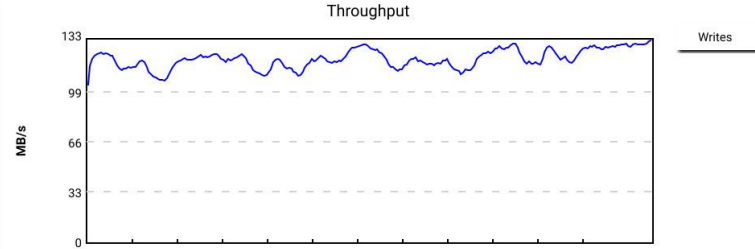
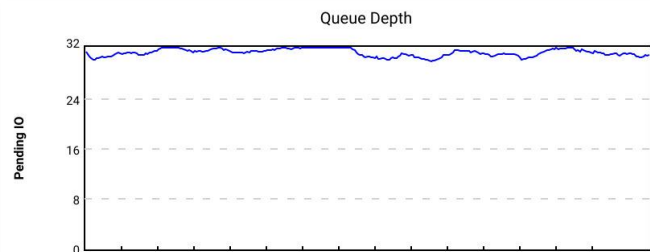
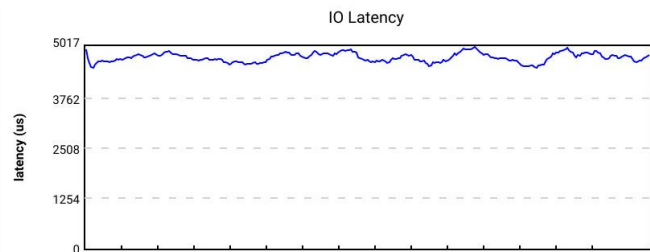
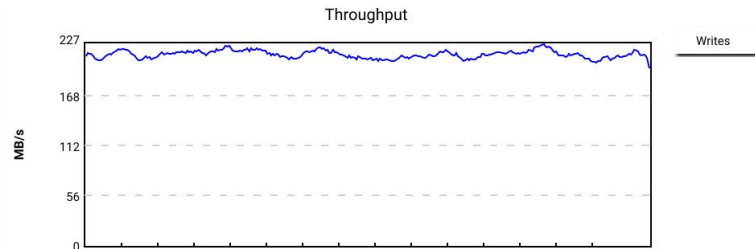
Attempts at reproduction

More faithful tweaking of setup and load

- Disable disk writeback cache to match customer setup
- Submit writes as O_DSYNC (observed blocking in fsync on IO completion in stacktraces)
- Disable rmapbt in SLE15-SP6

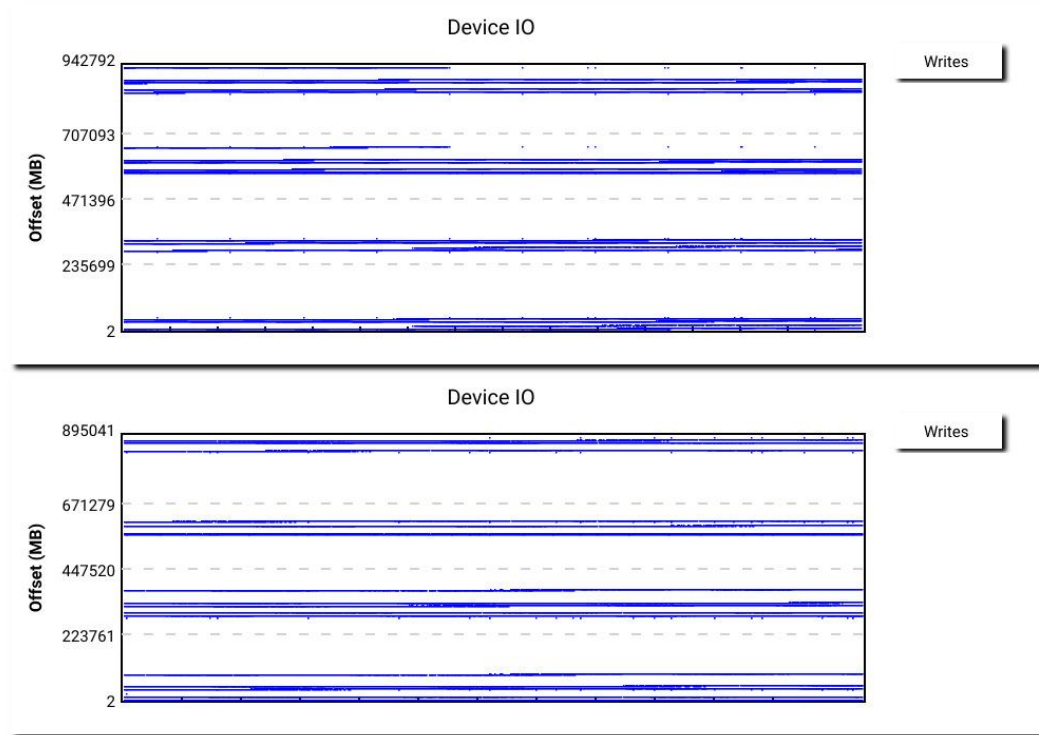
No significant difference between SLE12-SP5 and SLE15-SP6

Back to traces (iowatcher)



Back to traces (iowatcher)

DB2 uses userspace AIO implementation where threads pull work from pool



Yet more trace data

Still unclear how exactly DB2 behaves - so far seen only submitted IO

- Unclear how many files
- How threads are spread among files
- How much of IO is O_DSYNC
- Block traces captured only a few minutes of the whole restore run

Asked for full run blocktrace, XFS metadump after restore, syscall trace

```
trace-cmd record -e syscalls -f 'comm == "db2sysc"' -o trace-syscalls.dat
```

Systems with matched XFS feature set

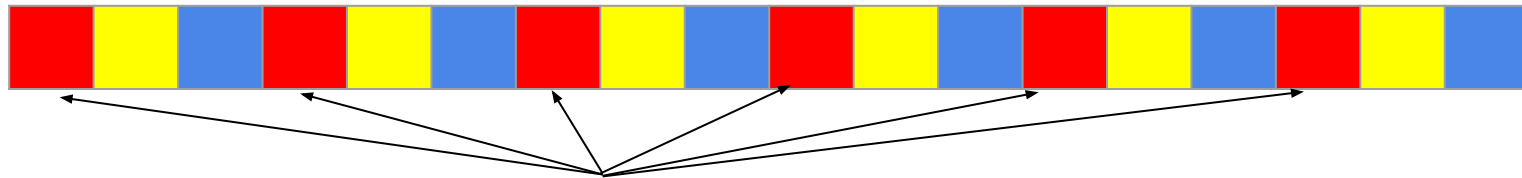
Data analysis

All IO is O_DSYNC direct IO

Files are preallocated with fallocate(2)

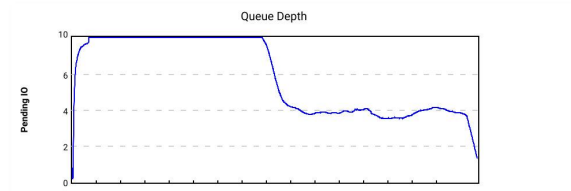
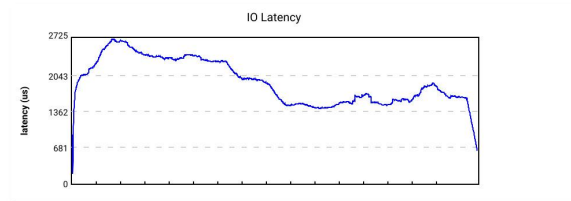
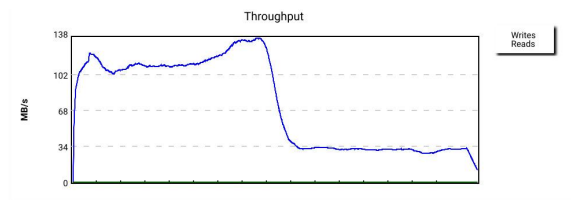
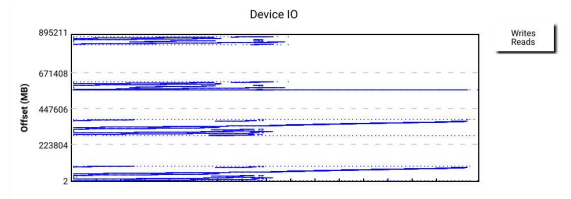
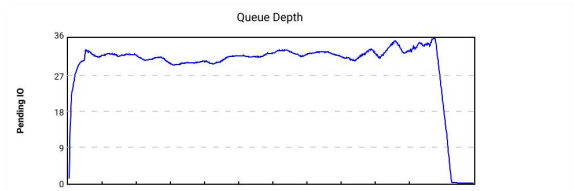
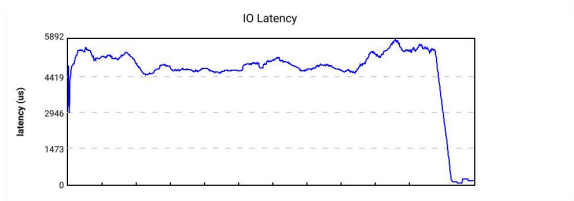
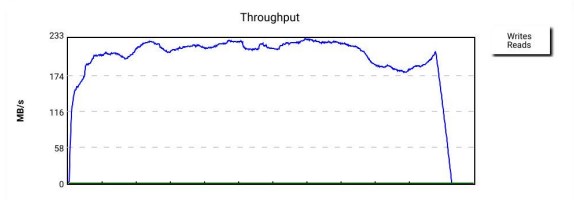
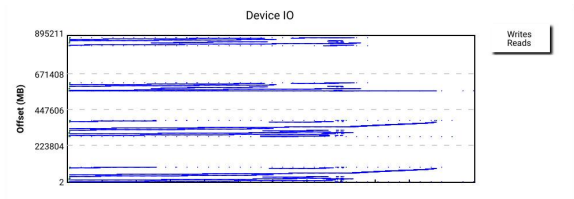
84 files restored, 2 large (about 30% of data), the rest has varying sizes

Very peculiar IO pattern where three tasks interleave sets of 3x6 32KB chunks



Run sequentially within task, unsynchronized with other two tasks

Data analysis



Simulator

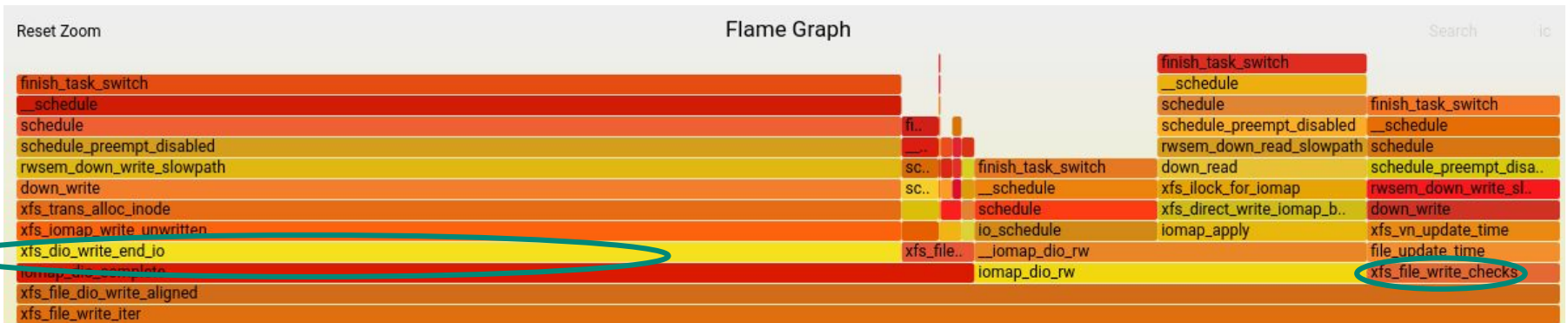
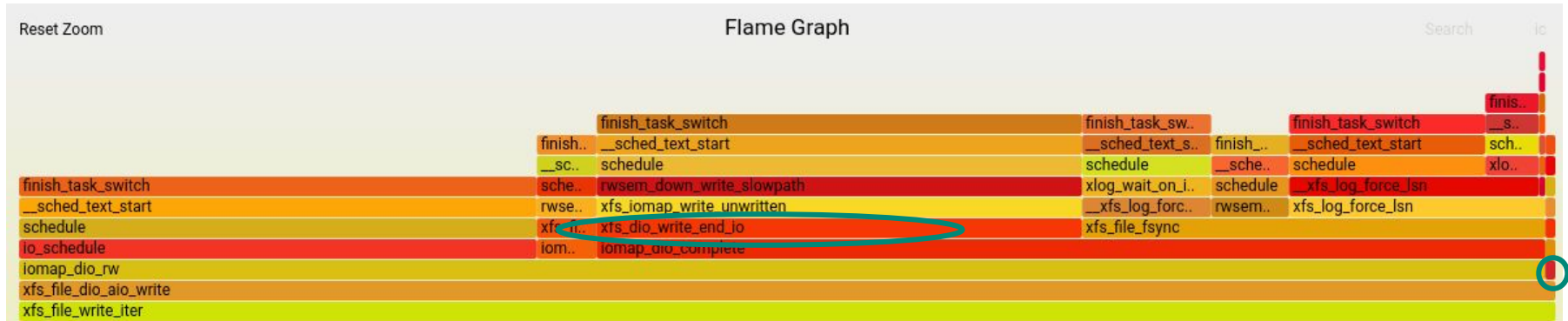
Wrote simulator replaying writes from the syscall trace (including from which task the write happened)

Small but stable difference between SLE12-SP5 and SLE15-SP6 (7m 24s vs 8m 2s) in a test VM with matched XFS features

Managed to bisect down the difference to SLE15-SP3 vs SLE15-SP4 timeframe

- 5.3 to 5.14 transition
- Then things got confusing

Flame graphs (SLE15-SP3 vs SLE15-SP4)



Flame graph analysis & bisection

Flame graphs indicate increase in XFS ILOCK rwsem contention

Only minor differences in XFS locking between SLE15 SP3 & SP4

More bisection to narrow down the cause

- Challenging to setup toolchain for these old kernel
- Upstream SLE15-SP3 base (5.3.19 and 5.4) is already slow ???
- Speedup in SLE15-SP3 happened during development due to 978088fdf42 ("locking/rwsem: Disable reader optimistic spinning")
- OK, but SLE15-SP4 doesn't do optimistic reader spinning either ???
- But it has 1a728dff855a ("locking/rwsem: Enable reader optimistic lock stealing")

The culprit

In SLE12-SP5 rwsem is a standard reader-writer sleeping lock

Over time it acquired writer lock spinning, reader lock spinning, later replaced with reader lock stealing

Trades fairness for latency and throughput wins

Changes to rwsem behavior resulted in more aggressive ILOCK readers (from `xfs_file_fsync()`) stalling ILOCK writers from IO submission and IO completion paths.

Confirmed that disabling lock stealing restores performance for the customer

How can we fix this? Reader lock stealing is a big win for some loads...

A glimpse of hope

Current (at the time) 6.16 kernel showed good performance



A glimmer of hope

Current (at the time) 6.16 kernel showed good performance

- Only because it had a data integrity bug caused by d279c80e0bac ("iomap: inline iomap_dio_bio_opflags()") resulting in xfs_file_fsync() not being called
- Fixed by 6b65028e2b51 ("iomap: Fix broken data integrity guarantees for O_SYNC writes")

Proper solution

`xfs_file_fsync()` can be optimized (with some tweaks to XFS logging and fsync tracking) to not acquire ILOCK

- Instead of remembering logged inode fields which need committing we can remember transaction IDs which need to be committed for the inode fsync / fdatsync to complete
- Worked with Dave Chinner on implementation
- Commit c91d38b57f2c ("xfs: rework datasync tracking and execution")



Conclusion

Conclusion

No simple takeaway, sometimes things are hard...

Various tools to get insight into how IO is done

- Iostat, blktrace, iowatcher, trace-cmd, BPF - flame graphs, ...

Don't give up!



Thank you

