# multichannel / io\_uring

Status Update within Samba

Stefan Metzmacher <metze@samba.org>

Samba Team / SerNet

2021-09-28

https://samba.org/~metze/presentations/2021/SDC/



## Check for Updates

- Check for an updated version of this presentation here:
- https://samba.org/~metze/presentations/2021/SDC/





### **Topics**

- ▶ What is SMB3 Multichannel?
- ▶ Updates in Samba 4.15
- ▶ What is io-uring?
- ▶ io-uring for Samba
- Performance research, prototyping and ideas
- Questions? Feedback!





- ▶ Multiple transport connections are bound to one logical connection
  - ► This allows using more than one network link
    - ► Good for performance
    - Good for availability reasons
  - ▶ Non TCP transports like RDMA (InfiniBand, RoCE, iWarp)
- ▶ All transport connections (channels) share the same CliendGUID
  - This is important for Samba
- An authenticated binding is done at the user session layer
  - SessionID, TreeID and FileID values are valid on all channels
- Available network interfaces are auto-negotiated
  - FSCTL\_QUERY\_NETWORK\_INTERFACE\_INFO interface list
  - ▶ IP (v4 or v6) addresses are returned together with:
    - ▶ Interface Index (which addresses belong to the same hardware)
    - Link speed
    - RSS and RDMA capabilities





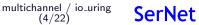
- ▶ Multiple transport connections are bound to one logical connection
  - This allows using more than one network link
    - Good for performance
    - Good for availability reasons
  - Non TCP transports like RDMA (InfiniBand, RoCE, iWarp)
- ▶ All transport connections (channels) share the same CliendGUID
  - ► This is important for Samba
- An authenticated binding is done at the user session layer
  - SessionID, TreeID and FileID values are valid on all channels
- Available network interfaces are auto-negotiated
  - ► FSCTL\_QUERY\_NETWORK\_INTERFACE\_INFO interface list
  - ► IP (v4 or v6) addresses are returned together with:
    - ▶ Interface Index (which addresses belong to the same hardware)
    - Link speed
    - RSS and RDMA capabilities





- ▶ Multiple transport connections are bound to one logical connection
  - ► This allows using more than one network link
    - ► Good for performance
  - Good for availability reasons
  - Non TCP transports like RDMA (InfiniBand, RoCE, iWarp)
- ▶ All transport connections (channels) share the same CliendGUID
  - ► This is important for Samba
- ▶ An authenticated binding is done at the user session layer
  - SessionID, TreeID and FileID values are valid on all channels
- Available network interfaces are auto-negotiated
  - ► FSCTL\_QUERY\_NETWORK\_INTERFACE\_INFO interface list
  - ▶ IP (v4 or v6) addresses are returned together with:
    - ▶ Interface Index (which addresses belong to the same hardware)
    - Link speed
    - RSS and RDMA capabilities





- ▶ Multiple transport connections are bound to one logical connection
  - ▶ This allows using more than one network link
    - Good for performance
  - Good for availability reasons
  - Non TCP transports like RDMA (InfiniBand, RoCE, iWarp)
- ▶ All transport connections (channels) share the same CliendGUID
  - This is important for Samba
- An authenticated binding is done at the user session layer
  - SessionID, TreeID and FileID values are valid on all channels
- Available network interfaces are auto-negotiated
  - FSCTL\_QUERY\_NETWORK\_INTERFACE\_INFO interface list
  - ▶ IP (v4 or v6) addresses are returned together with:
    - ▶ Interface Index (which addresses belong to the same hardware)
    - Link speed
    - RSS and RDMA capabilities



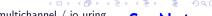




# Last Status Updates (SDC 2020 / SambaXP 2021)

- ▶ I gave a similar talk at the storage developer conference 2020:
  - See https://samba.org/~metze/presentations/2020/SDC/
  - ▶ It explains the milestones and design up to Samba 4.13 (in detail)
- ▶ I gave a similar talk at the SambaXP 2021:
  - See https://samba.org/~metze/presentations/2021/SambaXP/
  - ▶ It explains the milestones and updates up to Samba 4.15 (in detail)





#### Updates in Samba 4.15

- Automated regression tests are in place:
  - socket\_wrapper got basic fd-passing support(Bug #11899)
  - ▶ We added a lot more multichannel related regression tests
- ► The last missing features/bugs are fixed (Bug #14524)
  - ► The connection passing is fire and forget (Bug #14433)
  - ▶ Pending async operations are canceled (Bug #14449)
- ▶ 4.15 finally has "server multi channel support = yes"
  - ► We require support for TIOCOUTQ (Linux) or FIONWRITE (FreeBSD
  - ▶ We disable multichannel feature if the platform doesn't support this
    - ▶ See: Retries of Lease/Oplock Break Notifications (Bug #11898)





#### Updates in Samba 4.15

- Automated regression tests are in place:
  - socket\_wrapper got basic fd-passing support(Bug #11899)
  - ▶ We added a lot more multichannel related regression tests
- ▶ The last missing features/bugs are fixed (Bug #14524)
  - ► The connection passing is fire and forget (Bug #14433)
  - Pending async operations are canceled (Bug #14449)





#### Updates in Samba 4.15

- Automated regression tests are in place:
  - socket\_wrapper got basic fd-passing support(Bug #11899)
  - ▶ We added a lot more multichannel related regression tests
- ► The last missing features/bugs are fixed (Bug #14524)
  - ▶ The connection passing is fire and forget (Bug #14433)
  - ▶ Pending async operations are canceled (Bug #14449)
- ▶ 4.15 finally has "server multi channel support = yes"
  - ► We require support for TIOCOUTQ (Linux) or FIONWRITE (FreeBSD)
  - ▶ We disable multichannel feature if the platform doesn't support this
    - ► See: Retries of Lease/Oplock Break Notifications (Bug #11898)





# What is io-uring? (Part 1)

- Linux 5.1 introduced a new scalable AIO infrastructure
  - It's designed to avoid syscalls as much as possible
  - kernel and userspace share mmap'ed rings:
    - submission queue (SQ) ring buffer
    - completion queue (CQ) ring buffer
  - ► See "Ringing in a new asynchronous I/O API" on LWN.NET





# What is io-uring? (Part 1)

- ▶ Linux 5.1 introduced a new scalable AIO infrastructure
  - It's designed to avoid syscalls as much as possible
  - kernel and userspace share mmap'ed rings:
    - submission queue (SQ) ring buffer
    - completion queue (CQ) ring buffer
  - ► See "Ringing in a new asynchronous I/O API" on LWN.NET
- This can be nicely integrated with our async tevent model
  - It may delegate work to kernel threads
  - It seems to perform better compared to our userspace threadpool
  - It can also inline non-blocking operations





### io-uring for Samba (Part 1)

- ▶ Between userspace and filesystem (available from 5.1):
  - ► IORING\_OP\_READV, IORING\_OP\_WRITEV and IORING\_OP\_FSYNC
  - Supports buffered and direct io
- ▶ Between userspace and socket (and also filesystem) (from 5.8)
  - ► IORING\_OP\_SENDMSG, IORING\_OP\_RECVMSG
  - ▶ Improved MSG\_WAITALL support (5.12, backported to 5.11, 5.10)
  - ► IORING\_OP\_SPLICE, IORING\_OP\_TEE
  - ▶ Maybe using IORING\_SETUP\_SQPOLL or IOSQE\_ASYNC
- Path based syscalls with async impersonation (from 5.6)
  - ▶ IORING\_OP\_OPENAT2, IORING\_OP\_STATX
  - Using IORING\_REGISTER\_PERSONALITY for impersonation
  - ► IORING\_OP\_UNLINKAT, IORING\_OP\_RENAMEAT (from 5.10)
  - ► IORING\_OP\_MKDIRAT, IORING\_OP\_SYMLINKAT, IORING\_OP\_LINKAT (from 5.15)





### io-uring for Samba (Part 1)

- ▶ Between userspace and filesystem (available from 5.1):
  - ► IORING\_OP\_READV, IORING\_OP\_WRITEV and IORING\_OP\_FSYNC
  - Supports buffered and direct io
- ▶ Between userspace and socket (and also filesystem) (from 5.8)
  - ► IORING\_OP\_SENDMSG, IORING\_OP\_RECVMSG
  - ▶ Improved MSG\_WAITALL support (5.12, backported to 5.11, 5.10)
  - ► IORING\_OP\_SPLICE, IORING\_OP\_TEE
  - Maybe using IORING\_SETUP\_SQPOLL or IOSQE\_ASYNC
- Path based syscalls with async impersonation (from 5.6)
  - ► IORING\_OP\_OPENAT2, IORING\_OP\_STATX
  - Using IORING\_REGISTER\_PERSONALITY for impersonation
  - ► IORING\_OP\_UNLINKAT, IORING\_OP\_RENAMEAT (from 5.10)
  - ► IORING\_OP\_MKDIRAT, IORING\_OP\_SYMLINKAT, IORING\_OP\_LINKAT (from 5.15)





### io-uring for Samba (Part 1)

- ▶ Between userspace and filesystem (available from 5.1):
  - ► IORING\_OP\_READV, IORING\_OP\_WRITEV and IORING\_OP\_FSYNC
  - Supports buffered and direct io
- ▶ Between userspace and socket (and also filesystem) (from 5.8)
  - ► IORING\_OP\_SENDMSG, IORING\_OP\_RECVMSG
  - ▶ Improved MSG\_WAITALL support (5.12, backported to 5.11, 5.10)
  - ► IORING\_OP\_SPLICE, IORING\_OP\_TEE
  - Maybe using IORING\_SETUP\_SQPOLL or IOSQE\_ASYNC
- ▶ Path based syscalls with async impersonation (from 5.6)
  - IORING\_OP\_OPENAT2, IORING\_OP\_STATX
  - ▶ Using IORING\_REGISTER\_PERSONALITY for impersonation
  - ► IORING\_OP\_UNLINKAT, IORING\_OP\_RENAMEAT (from 5.10)
  - ► IORING\_OP\_MKDIRAT, IORING\_OP\_SYMLINKAT, IORING\_OP\_LINKAT (from 5.15)





## io-uring for Samba (Part 2)

#### IORING\_FEAT\_NATIVE\_WORKERS (from 5.12)

- ▶ In the kernel...
  - ► The io-uring kernel threads are clone()'ed from the userspace thread
  - They just appear to be blocked in a syscall and never return
  - ▶ This makes the accounting in the kernel much saner
  - ▶ Allows a lot of restrictions to be relaxed in the kernel
- For admins and userspace developers..
  - ► They are no longer 'io\_wqe\_work' kernel threads
  - 'top' shows them as part of the userspace process ('H' shows them)
  - ► They are now visible in containers
  - 'pstree -a -t -p' is very useful to see them
  - ▶ They are shown as iou-wrk-1234, for a task with pid/tid 1234





### io-uring for Samba (Part 2)

#### IORING\_FEAT\_NATIVE\_WORKERS (from 5.12)

- ▶ In the kernel...
  - ► The io-uring kernel threads are clone()'ed from the userspace thread
  - ▶ They just appear to be blocked in a syscall and never return
  - ▶ This makes the accounting in the kernel much saner
  - ▶ Allows a lot of restrictions to be relaxed in the kernel
- For admins and userspace developers...
  - They are no longer 'io\_wqe\_work' kernel threads
  - 'top' shows them as part of the userspace process ('H' shows them)
  - ► They are now visible in containers
  - 'pstree -a -t -p' is very useful to see them
  - ▶ They are shown as iou-wrk-1234, for a task with pid/tid 1234





# vfs\_io\_uring in Samba 4.12 (2020)

- ▶ With Samba 4.12 we added "io\_uring" vfs module
  - ► For now it only implements SMB\_VFS\_PREAD,PWRITE,FSYNC\_SEND/RECV
  - ▶ It has less overhead than our pthreadpool default implementations
  - ▶ I was able to speed up a smbclient 'get largefile /dev/null'
    - Using against smbd on loopback
    - ▶ The speed changes from 2.2GBytes/s to 2.7GBytes/s
- The improvement only happens by avoiding context switches
  - But the data copying still happens:
    - From/to a userspace buffer to/from the filesystem/page cache
  - ▶ The data path between userspace and socket is completely unchanged
  - For both cases the cpu is mostly busy with memcpy





# vfs\_io\_uring in Samba 4.12 (2020)

- ▶ With Samba 4.12 we added "io\_uring" vfs module
  - ► For now it only implements SMB\_VFS\_PREAD,PWRITE,FSYNC\_SEND/RECV
  - ▶ It has less overhead than our pthreadpool default implementations
  - ▶ I was able to speed up a smbclient 'get largefile /dev/null'
    - Using against smbd on loopback
    - ► The speed changes from 2.2GBytes/s to 2.7GBytes/s
- ▶ The improvement only happens by avoiding context switches
  - But the data copying still happens:
    - ► From/to a userspace buffer to/from the filesystem/page cache
  - ▶ The data path between userspace and socket is completely unchanged
  - For both cases the cpu is mostly busy with memcpy





# Performance research (SMB2 Read)

- ▶ In October 2020 I was able to do some performance research
  - ▶ With 100GBit/s interfaces and two NUMA nodes per server.
- ▶ At that time I focussed on the SMB2 Read performance only
  - We had limited time on the given hardware
  - We mainly tested with fio.exe on a Windows client
  - ▶ Linux kernel 5.8.12 on the server
- More verbose details can be found here:
  - https://lists.samba.org/archive/samba-technical/2020-October/135856.html





# Performance research (SMB2 Read)

- ▶ In October 2020 I was able to do some performance research
  - ▶ With 100GBit/s interfaces and two NUMA nodes per server.
- ▶ At that time I focussed on the SMB2 Read performance only
  - We had limited time on the given hardware
  - We mainly tested with fio.exe on a Windows client
  - ▶ Linux kernel 5.8.12 on the server
- More verbose details can be found here:
  - https://lists.samba.org/archive/samba-technical/2020-October/135856.htm





# Performance research (SMB2 Read)

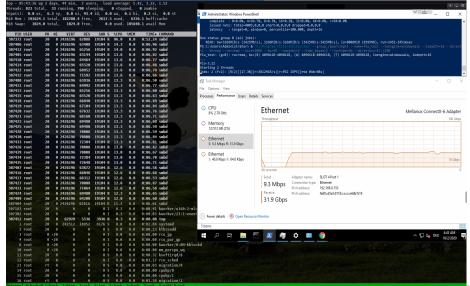
- ▶ In October 2020 I was able to do some performance research
  - ▶ With 100GBit/s interfaces and two NUMA nodes per server.
- ▶ At that time I focussed on the SMB2 Read performance only
  - ▶ We had limited time on the given hardware
  - We mainly tested with fio.exe on a Windows client
  - Linux kernel 5.8.12 on the server
- More verbose details can be found here:
  - https://lists.samba.org/archive/samba-technical/2020-October/135856.html





# Performance with MultiChannel, sendmsg()

4 connections, ~3.8 GBytes/s, bound by >500% cpu in total, sendmsg() takes up to 0.5 msecs



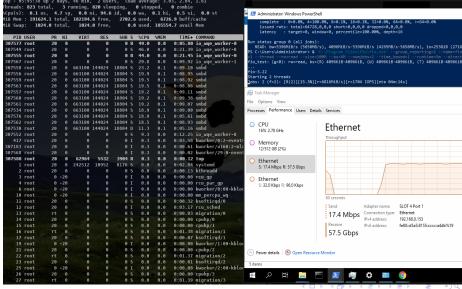


multichannel / io\_uring (12/22)

**SerNet** 

# IORING\_OP\_SENDMSG (Part1)

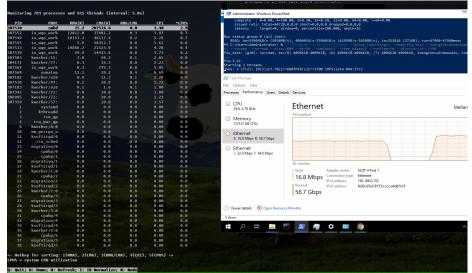
4 connections, "6.8 GBytes/s, smbd only uses "11% cpu, (io\_wqe\_work "50% cpu) per connection, we still use >300% cpu in total top - 05:45:38 up 2 days, 46 min, 2 users, load average: 3.03, 2.84, 1.61





### IORING\_OP\_SENDMSG (Part2)

The results vary havily depending on the NUMA bouncing, between 5.0 GBytes/s and 7.6 GBytes/s

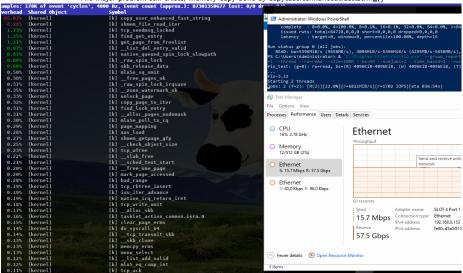






### IORING\_OP\_SENDMSG (Part3)

The major problem still exists, memory copy done by copy\_user\_enhanced\_fast\_string()

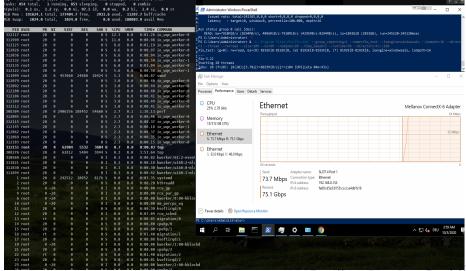




# IORING\_OP\_SENDMSG + IORING\_OP\_SPLICE (Part1)

16 connections, ~8.9 GBytes/s, smbd ~5% cpu, (io\_wqe\_work 3%-12% cpu filesystem->pipe->socket), only ~100% cpu in total.

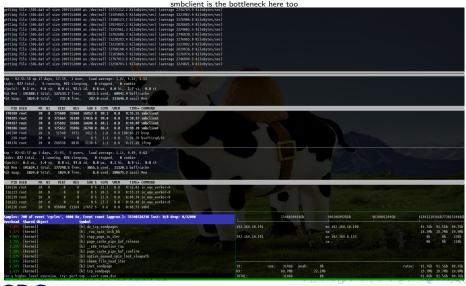
 $The \ Windows \ client \ was \ still \ the \ bottleneck \ with \ "Set-SmbClientConfiguration - ConnectionCountPerRssNetworkInterface \ 16"$ 





# smbclient IORING\_OP\_SENDMSG/SPLICE (network)

4 connections, ~11 GBytes/s, smbd 8.6% cpu, with 4 io\_wqe\_work threads (pipe to socket) at ~20% cpu each.





# smbclient IORING\_OP\_SENDMSG/SPLICE (loopback)

8 connections, ~22 GBytes/s, smbd 22% cpu, with 4 io\_wqe\_work threads (pipe to socket) at ~22% cpu each.

smbclient is the bottleneck here too, it triggers the memory copy done by copy\_user\_enhanced\_fast\_string() top - 84:00:58 up 4 days, 23:02, 6 users, load average: 9.15, 3.56, 1.44 file \586.dat of size 2897152888 as /dev/null (3875874.6 KiloBytes/sec) (average 2888881.8 KiloBytes/sec) Tasks: 917 total, 14 running, 983 sleeping, 8 stopped, 8 zombie etting file \506.dat of size 2097152000 as /dev/null (2942520.3 KiloBytes/sec) (average 2943679.6 KiloBytes/sec) etting file \586.dat of size 2897152888 as /dev/null (2719787.2 KiloBytes/sec) (average 2841637.3 KiloBytes/sec) MCpu(s): 0.3 us, 11.2 sy, 0.0 ni, 86.1 id, 0.0 wa, 0.2 hi, 2.1 si, 0.0 st etting file \586.dat of size 2897152888 as /dev/null (2951888.2 KiloBytes/sec) (average 2879437.6 KiloBytes/sec) MiB Mem : 191624.1 total, 176925.4 free, 3316.7 used, 11382.8 buff/cache etting file \586.dat of size 2897152888 as /dev/null (2881641.2 KiloBytes/sec) (average 2739178.8 KiloBytes/sec) MiB Swap: 1024.0 total, 1024.0 free, A A mend 188483 7 avail Non SHR S 1/CPU 1/MEM etting file \506.dat of size 2097152000 as /dev/null (3117198.9 KiloBytes/sec) (average 2090262.3 KiloBytes/sec) petting file \586 dat of size 2897152888 as /dev/null (3867618.6 KiloRytes/sec) (average 2944358.1 KiloRytes/sec) 322765 root petting file \586 dat of size 2897152888 as /dev/pull (3898335.4 KiloRytes/sec) (average 2741473.6 KiloRytes/sec) 322768 root petting file \586.dat of size 2897152888 as /dev/mull (2741632.8 KiloRytes/sec) (average 2848912.6 KiloRytes/sec) 322762 root petting file \586.dat of size 2897152888 as /dev/mull (3882932.1 KiloRytes/sec) (average 2888254.5 KiloRytes/sec) 322761 root 322766 root 322759 roof etting file \506.dat of size 2097152000 as /dev/null (3088939.0 KiloBytes/sec) (average 2091536.4 KiloBytes/sec) etting file \586.dat of size 2897152888 as /dev/null (2515978.2 KiloRytes/sec) (average 2731748.8 KiloRytes/sec 322782 roof etting file \506.dat of size 2097152000 as /dev/null (2171791.9 KiloBytes/sec) (average 2709204.0 KiloBytes/sec) 322827 root etting file \586.dat of size 2897152888 as /dev/null (2921548.2 KiloBytes/sec) (average 2944283.8 KiloBytes/sec) etting file \586.dat of size 2897152888 as /dev/null (3893655.1 KiloBytes/sec) (average 2743728.7 KiloBytes/sec) etting file \506.dat of size 2097152000 as /dev/null (3093655.1 KiloBytes/sec) (average 2042525.3 KiloBytes/sec etting file \586.dat of size 2097152000 as /dev/null (3007341.7 KiloBytes/sec) (average 2031000.4 KiloBytes/sec 322796 root etting file \586.dat of size 2097152000 as /dev/null (3107738.5 KiloBytes/sec) (average 2060079.4 KiloBytes/sec etting file \586.dat of size 2897152888 as /dev/null (3136293.6 KiloBytes/sec) (average 2893872.3 KiloBytes/sec) 222222 root etting file \586.dat of size 2897152888 as /dev/null (2752887.8 KiloBytes/sec) (average 2731898.3 KiloBytes/sec) 322818 root etting file \506.dat of size 2097152000 as /dev/null (3004336.9 KiloBytes/sec) (average 2045095.8 KiloBytes/sec) 318818 root petting file \586.dat of size 2897152888 as /dev/null (2745388.8 KiloBytes/sec) (average 2789462.2 KiloBytes/sec) 322533 root petting file \586.dat of size 2097152000 as /dev/null (3117198.9 KiloBytes/sec) (average 2746078.8 KiloBytes/sec) 322854 root metting file \506.dat of size 2097152000 as /dev/null (3117198.9 KiloBytes/sec) (average 2044253.7 KiloBytes/sec) 322842 root metting file \50G.dat of size 2097152000 as /dev/null (2563203.7 KiloBytes/sec) (average 2578659.5 KiloBytes/sec) 322851 root getting file \506.dat of size 2097152000 as /dev/null (2519064.9 KiloBytes/sec) (average 255651.4 KiloBytes/sec) 322868 root getting file \506.dat of size 2097152000 as /dev/null (3093655.1 KiloBytes/sec) (average 2094340.3 KiloBytes/sec) 322862 root etting file \50G.dat of size 2097152000 as /dev/null (2020728.9 KiloBytes/sec) (average 2732506.5 KiloBytes/sec) 1:49.89 perf etting file \50G.dat of size 2097152000 as /dev/null (2771312.2 KiloBytes/sec) (average 2709897.3 KiloBytes/sec) etting file \506.dat of size 2097152000 as /dev/null (3131498.0 KiloBytes/sec) (average 2846041.8 KiloBytes/sec) 8:82.77 io wae worker-8 etting file \506.dat of size 2097152000 as /dev/null (3131498.0 KiloBytes/sec) (average 2748470.0 KiloBytes/sec) 322848 root 8:82.52 io wge worker-8 etting file \506.dat of size 2097152000 as /dev/null (2595690.4 KiloBytes/sec) (average 2942472.7 KiloBytes/sec) 322865 root 8:82.68 io wae worker-8 etting file \506.dat of size 2097152000 as /dev/null (3038575.2 KiloBytes/sec) (average 2957176.0 KiloBytes/sec) 322868 root etting file \50G.dat of size 2097152000 as /dev/null (2976743.8 KiloBytes/sec) (average 2879300.8 KiloBytes/sec) 322887 root 8:82.57 io wae worker-8 etting file \50G.dat of size 2097152000 as /dev/null (3038575.2 KiloBytes/sec) (average 2895262.7 KiloBytes/sec) 20 8.8 8:82.58 io wae worker-8 etting file \506.dat of size 2097152000 as /dev/null (2024027.2 KiloBytes/sec) (average 2733199.6 KiloBytes/sec) 3.6 8.8 8:82.33 io wge worker-8 8 S 3.6 8.8 8:82.52 io wge worker-8 amples: 30M of event 'cycles', 1888 Hz, Event count (approx.): 526765589529 lost: 6/8 drop: 0/8 15755379286b 47266148166b verhead Shared Object [kernel] [k] native queued spin lock slowpath [k] raw spin lock bh [k] raw spin lock [k] copy page to iter Ikl skh release data 2264268 neak: 1816b 1816b [k] check object size



### More loopback testing on brand new hardware

- Recently I re-did the loopback read tests IORING\_OP\_SENDMSG/SPLICE (from /dev/shm/)
  - ▶ 1 connection, ~10-13 GBytes/s, smbd 7% cpu, with 4 iou-wrk threads at 7%-50% cpu.
  - ▶ 4 connections, 24-30 GBytes/s, smbd 18% cpu, with 16 iou-wrk threads at 3%-35% cpu.
- ► I also implemented SMB2 writes with IORING\_OP\_RECVMSG/SPLICE (tested to /dev/null)
  - ▶ 1 connection, ~7-8 GBytes/s, smbd 5% cpu, with 3 io-wrk threads at 1%-20% cpu.
  - ▶ 4 connections, ~10 GBytes/s, smbd 15% cpu with 12 io-wrk threads at 1%-20% cpu.
- I tested with a Linux Kernel 5.13
  - ▶ In both cases the bottleneck is clearly on the smbclient side
  - ▶ We could apply similar changes to smbclient and add true multichannel support
    - It seems that the filesystem->pipe->socket path is much better optimized



#### More loopback testing on brand new hardware

- Recently I re-did the loopback read tests IORING\_OP\_SENDMSG/SPLICE (from /dev/shm/)
  - ▶ 1 connection, ~10-13 GBytes/s, smbd 7% cpu, with 4 iou-wrk threads at 7%-50% cpu.
    - ▶ 4 connections, 24-30 GBytes/s, smbd 18% cpu, with 16 iou-wrk threads at 3%-35% cpu.
- I also implemented SMB2 writes with IORING\_OP\_RECVMSG/SPLICE (tested to /dev/null)
  - ▶ 1 connection, ~7-8 GBytes/s, smbd 5% cpu, with 3 io-wrk threads at 1%-20% cpu.
  - ▶ 4 connections, ~10 GBytes/s, smbd 15% cpu, with 12 io-wrk threads at 1%-20% cpu.

Stefan Metzmacher

- ▶ I tested with a Linux Kernel 5.13
  - ▶ In both cases the bottleneck is clearly on the smbclient side
  - ▶ We could apply similar changes to smbclient and add true multichannel support
  - ▶ It seems that the filesystem->pipe->socket path is much better optimized



### More loopback testing on brand new hardware

- Recently I re-did the loopback read tests IORING\_OP\_SENDMSG/SPLICE (from /dev/shm/)
  - ► 1 connection, ~10-13 GBytes/s, smbd 7% cpu,
    - with 4 iou-wrk threads at 7%-50% cpu.
    - ▶ 4 connections, 24-30 GBytes/s, smbd 18% cpu, with 16 iou-wrk threads at 3%-35% cpu.
- ► I also implemented SMB2 writes with IORING\_OP\_RECVMSG/SPLICE (tested to /dev/null)
  - ▶ 1 connection, ~7-8 GBytes/s, smbd 5% cpu, with 3 io-wrk threads at 1%-20% cpu.
  - ▶ 4 connections, ~10 GBytes/s, smbd 15% cpu, with 12 io-wrk threads at 1%-20% cpu.
- ▶ I tested with a Linux Kernel 5.13
  - ▶ In both cases the bottleneck is clearly on the smbclient side
  - We could apply similar changes to smbclient and add true multichannel support
  - It seems that the filesystem->pipe->socket path is much better optimized



# Improvements for transfers with SMB3 signing

- ▶ Samba 4.15 has support for AES-128-GMAC signing:
  - ▶ This is also available in recent Windows versions
  - ▶ It's based on AES-128-GCM (but only with authentication data)
  - ► The gnutls library is able to provide:
    - ~6 GBytes/s for AES-128-GCM
    - ~10 GBytes/s for AES-128-GMAC
- ► For SMB3 signing/encryption we use
  - ► IORING\_OP\_SPLICE from a file into a (splice)pipe
  - ► IORING\_OP\_TEE from the (splice)pipe to a 2nd (tee)pipe
  - ▶ IORING\_OP\_READ from the (tee)pipe into a userspace buffer
    - (vmsplice might work even better)
  - ▶ The userspace buffer is only used to calculate the signing signature
  - ▶ IORING\_OP\_SENDMSG and IORING\_OP\_SPLICE are used in order to avoid a copy back to the kernel
- ► For a SMB2 read test I removed the signing check in smbclient:
  - The performance changed from ~3 GBytes/s before
  - ► To ~5 GBytes/s using the IORING\_OP\_TEE trick
    - ▶ With smbclient still being the bootleneck at 100% cpu



# Improvements for transfers with SMB3 signing

- ► Samba 4.15 has support for AES-128-GMAC signing:
  - ▶ This is also available in recent Windows versions
  - ▶ It's based on AES-128-GCM (but only with authentication data)
  - ► The gnutls library is able to provide:
    - ~6 GBytes/s for AES-128-GCM
    - ~10 GBytes/s for AES-128-GMAC
- ► For SMB3 signing/encryption we use:
  - ► IORING\_OP\_SPLICE from a file into a (splice)pipe
  - ► IORING\_OP\_TEE from the (splice)pipe to a 2nd (tee)pipe
  - ► IORING\_OP\_READ from the (tee)pipe into a userspace buffer
    - (vmsplice might work even better)
  - ▶ The userspace buffer is only used to calculate the signing signature
  - IORING\_OP\_SENDMSG and IORING\_OP\_SPLICE are used in order to avoid a copy back to the kernel
- ► For a SMB2 read test I removed the signing check in smbclient:
  - ▶ The performance changed from ~3 GBytes/s before
  - ► To ~5 GBytes/s using the IORING\_OP\_TEE trick
    - ▶ With smbclient still being the bootleneck at 100% cpu





# Improvements for transfers with SMB3 signing

- ► Samba 4.15 has support for AES-128-GMAC signing:
  - ► This is also available in recent Windows versions
  - ▶ It's based on AES-128-GCM (but only with authentication data)
  - ► The gnutls library is able to provide:
    - ▶ ~6 GBytes/s for AES-128-GCM
    - ~10 GBytes/s for AES-128-GMAC
- ► For SMB3 signing/encryption we use:
  - ► IORING\_OP\_SPLICE from a file into a (splice)pipe
  - ► IORING\_OP\_TEE from the (splice)pipe to a 2nd (tee)pipe
  - ► IORING\_OP\_READ from the (tee)pipe into a userspace buffer
    - (vmsplice might work even better)
  - ▶ The userspace buffer is only used to calculate the signing signature
  - IORING\_OP\_SENDMSG and IORING\_OP\_SPLICE are used in order to avoid a copy back to the kernel
- ► For a SMB2 read test I removed the signing check in smbclient:
  - ▶ The performance changed from ~3 GBytes/s before
    - To ~5 GBytes/s using the IORING\_OP\_TEE trick
      With smbclient still being the bootleneck at 100% cpu



#### Future Improvements

- recvmsg and splice deliver partial SMB packets to userspace
  - ▶ I tested with AF\_KCM (Kernel Connection Multiplexor) and an eBPF helper
  - ▶ But MSG\_WAITALL is the much simpler and faster solution
  - I also prototyped a SPLICE\_F\_WAITALL
  - ▶ eBPF support in io-uring would also be great for optimizations
- It also seems that socket->pipe->filesystems
  - Does not implement zero copy for all cases
  - Maybe it's possible to optimize this in future
- ▶ In the end SMB-Direct will also be able to reduce overhead
  - My smbdirect driver is still work in progress..
  - With the IORING\_FEAT\_NATIVE\_WORKERS feature it will be possible glue it to IORING\_OP\_SENDMSG





#### Future Improvements

- recvmsg and splice deliver partial SMB packets to userspace
  - ▶ I tested with AF\_KCM (Kernel Connection Multiplexor) and an eBPF helper
  - ▶ But MSG\_WAITALL is the much simpler and faster solution
  - I also prototyped a SPLICE\_F\_WAITALL
  - ▶ eBPF support in io-uring would also be great for optimizations
- It also seems that socket->pipe->filesystem:
  - Does not implement zero copy for all cases
  - Maybe it's possible to optimize this in future
- In the end SMB-Direct will also be able to reduce overhead
  - My smbdirect driver is still work in progress..
  - With the IORING\_FEAT\_NATIVE\_WORKERS feature it will be possible glue it to IORING\_OP\_SENDMSG





#### Future Improvements

- recvmsg and splice deliver partial SMB packets to userspace
  - ▶ I tested with AF\_KCM (Kernel Connection Multiplexor) and an eBPF helper
  - ▶ But MSG\_WAITALL is the much simpler and faster solution
  - ▶ I also prototyped a SPLICE\_F\_WAITALL
  - eBPF support in io-uring would also be great for optimizations
- ▶ It also seems that socket->pipe->filesystem:
  - ▶ Does not implement zero copy for all cases
  - Maybe it's possible to optimize this in future
- ▶ In the end SMB-Direct will also be able to reduce overhead
  - My smbdirect driver is still work in progress...
  - ▶ With the IORING\_FEAT\_NATIVE\_WORKERS feature it will be possible glue it to IORING\_OP\_SENDMSG





#### Questions? Feedback!

► Feedback regarding real world testing would be great!

Stefan Metzmacher

- ▶ Stefan Metzmacher, metze@samba.org
- ► https://www.sernet.com
- https://samba.plus

Slides: https://samba.org/~metze/presentations/2021/SDC/



