

multichannel / io_uring

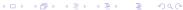
Status Update within Samba

Stefan Metzmacher <metze@samba.org>

Samba Team / SerNet

2021-05-05

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



Check for Updates

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

(draft)



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 FleID values are valid on all channels
- Available network interfaces are auto-negotiated
 - ► FSCTL QUERK_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

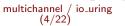






- Multiple transport connections are bound to one logical connection
 - ► This allows using more than one network link
 - Good for performance
 - Good for availability reasons
 - ► Non TCP transports like RDMA (InfiniBand, RoCE, iWarp)
- ▶ All transport connections (channels) share the same CliendGUID
 - This is important for Samba
- An authenticated binding is done at the user session layer
 - SessionID, TreeID and Fle Q 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

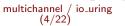






- Multiple transport connections are bound to one logical connection
 - ▶ This allows using more than one network link
 - Good for performance
 - Good for availability reasons
 - ► Non TCP transports like RDMA (InfiniBand, RoCE, iWarp)
- ▶ All transport connections (channels) share the same CliendGUID
 - 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 QUENT_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







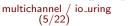
- ▶ IO ordering is important for multichannel
 - Requests can get lost between client and server
 - Responses can get lost between server and client
 - ► The client isn't able to know the difference
 - Replays contain the REPLAY flag in the SMB2 header
 - ► FILE_NOT_AVAILABLE indicates "please retry" to the client
 - Windows returns ACCESS_DENIED in some cases instead
 - In other cases Windows ignores a replay and deadlocks the client
 - ▶ I need to discuss this with Microsoft
 - See: Samba Bug #14449
- State changing operation, beta replay detection
 - They need to execute only-once
 - ► SMB2 Create uses a CreateGUID
 - SMB2 Lock uses an array with sequence numbers
 - Windows only supports this on resilient and persistent handles
 - ► Future Windows versions are supposed to fix that





- ▶ IO ordering is important for multichannel
 - Requests can get lost between client and server
 - Responses can get lost between server and client
 - ► The client isn't able to know the difference
 - Replays contain the REPLAY flag in the SMB2 header
 - ► FILE_NOT_AVAILABLE indicates "please retry" to the client
 - Windows returns ACCESS_DENIED in some cases instead
 - In other cases Windows ignores a replay and deadlocks the client
 - ► I need to discuss this with Microsoft
 - See: Samba Bug #14449
- State changing operations need replay detection
 - They need to execute only-once
 - SMB2 Create uses a CreateGUID
 - SMB2 Lock uses an array with sequence numbers
 - Windows only supports this on resilient and persistent handles
 - Future Windows versions are supposed to fix that







- Write/Set operations only need a barrier
 - ▶ An epoch number is incremented on each channel failure
 - ▶ The current epoch number is part of each request
 - ▶ The server remembers the last seen epoch number
 - ▶ Non-REPLAY requests with stale epoch fail
 - ▶ REPLAY requests fail, when there are pending older epoch numbers
- Read/Get operations can be replayed safely
- Lease/Oplock break notifications should be retried
 - Break notifications wait for transport acks
 - On channel failures they are retried on other channels
 - Windows doesn't retry for oplocks, only leases





- Write/Set operations only need a barrier
 - ▶ An epoch number is incremented on each channel failure
 - ► The current epoch number is part of each request
 - ▶ The server remembers the last seen epoch number
 - Non-REPLAY requests with stale epoch fail
 - ▶ REPLAY requests fail, when there are pending older epoch numbers
- Read/Get operations can be replayed safely
- Lease/Oplock break notifications should be retried
 - Break notifications wait for transport acks
 - On channel failures they are retried on other channels
 - Windows doesn't retry for oplocks, only leases





- Write/Set operations only need a barrier
 - ▶ An epoch number is incremented on each channel failure
 - ► The current epoch number is part of each request
 - ▶ The server remembers the last seen epoch number
 - ▶ Non-REPLAY requests with stale epoch fail
 - ▶ REPLAY requests fail, when there are pending older epoch numbers
- Read/Get operations can be replayed safely
- ► Lease/Oplock break notifications should be retried
 - Break notifications wait for transport acks
 - On channel failures they are retried on other channels
 - Windows doesn't retry for oplocks, only leases





Last Status Update SDC 2020

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





- 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 (Jug #14433)
 - ▶ Pending async operations are canceled (Bug 14449)
- ▶ 4.15 will hopefully have "server multi channel support = yes"
 - ► Currently it's still off by default, but may change before 4.15.0rc1
 - ► We require support for TIQCOUTQ (Linux) or FIONWRITE (FreeBSD)
 - We disable multicuannel feature if the platform doesn't support this
 - See Retries of Lease/Oplock Break Notifications (Bug #11898)
- I have unofficial backport for older branches
 - ► SerNet's SAMBA+ 4.14 includes the patches
 - "server multi channel support = no" is still the default



- ► 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 will hopefully have "server multi-channel support = yes"
 - ► Currently it's still off by default, but may change before 4.15.0rc1
 - We require support for TIQQQUTQ (Linux) or FIONWRITE (FreeBSD
 - We disable multituannel feature if the platform doesn't support this
 - ► Seet Retries of Lease/Oplock Break Notifications (Bug #11898)
- ▶ I have unoficial backport for older branches
 - ► SerNet's SMMBA+ 4.14 includes the patches
 - "server multi channel support = no" is still the default



- 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 will hopefully have "server multi channel support = yes"
 - Currently it's still off by default, but may change before 4.15.0rc1
 - ► 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)
- ▶ I have unoficial backport for older branches
 - ► SerNet's SMMBA+ 4.14 includes the patches
 - "server multi channel support = no" is still the default



- 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 will hopefully have "server multi channel support = yes"
 - Currently it's still off by default, but may change before 4.15.0rc1
 - ► 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)
- ▶ I have unofficial backport for older branches
 - ► SerNet's SAMBA+ 4.14 includes the patches
 - "server multi channel support = no" is still the default



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 threadpoor
 - ▶ It can also inline non-blocking operations





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

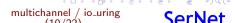




io-uring for Samba (Part 1)

- ▶ 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, backport to 5.11, 5.10)
 - ▶ IORING_OP_SPLICE, IORING_OP_TEE
 - Maybe using IORING_SETUP_SQPOLL or IOSQE_ASYNC
- ▶ Path based syscalls with as vivc impersonation (from 5.6)
 - ► IORING OF OPENAT2, IORING_OP_STATX
 - ▶ Using ORING REGISTER_PERSONALITY for impersonation
 - ► IORING OP UNLINKAT, IORING OP RENAMEAT (from 5.10)





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, backport 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)







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
 - Most likely to backported to the 5.10 LTS kernel
- For admins and userspace developers...
 - 'top' shows them as part of the userspace process ('H' shows them)
 - They are now visible in containers
 - b 'pstree -a b' is very useful to see them
 - gdb may show working messages
 - "warning: Architecture rejected target-supplied description"
 - ▶ But N seems they can be ignored and will be fixed soon



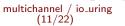


io-uring for Samba (Part 2)

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
 - Most likely to backported to the 5.10 LTS kernel
- For admins and userspace developers...
 - '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
 - gdb may show worrying messages:
 - "warning: Architecture rejected target-supplied description"
 - But it seems they can be ignored and will be fixed soon







Performance research (SMB2 Read)

- Last October I was able to do some performance research
 - ▶ DDN was so kind to sponsor about a week of research on real world hardware
 - With 100GBit/s interfaces and two NUMA nodes per server.





Performance research (SMB2 Read)

- ▶ Last October I was able to do some performance research
 - DDN was so kind to sponsor about a week of research on real world hardware
 - With 100GBit/s interfaces and two NUMA nodes per server.
- ▶ 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://lives.samba.org/archive/samba-technical/2020-October/135856.html

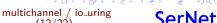




Performance research (SMB2 Read)

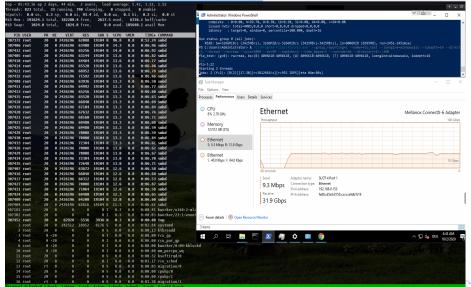
- Last October I was able to do some performance research
 - ▶ DDN was so kind to sponsor about a week of research on real world hardware
 - With 100GBit/s interfaces and two NUMA nodes per server.
- ▶ I focussed on the SMB2 Read performance only
 - We had limited time on the given hardware
 - We mainly 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





IORING_OP_SENDMSG prototyped (Part1)

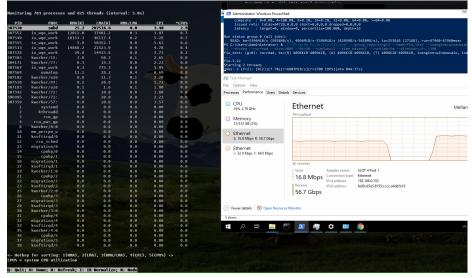
4 connections, "6.8 GBytes/s, smbd only uses "11% cpu, (io_wqe_work "50% cpu) per connection, we still use >300% cpu in total op - 05:45:38 up 2 days, 46 min, 2 users, load average: 3.03, 2.84, 1.61 Threads: 823 total, 3 running, 820 sleeping, 0 stopped, 0 zombie Cpu(s): 0.1 us, 4.7 sy, 0.0 ni, 94.6 id, 0.0 wa, 0.1 hi, 0.5 si, 0.0 st 2 Administrator: Windows PowerShell iB Mem : 191624.1 total, 182194.6 free, 2702.6 used, 6726.9 buff/cache complete : 0-0.0%, 4-100.0%, 8-0.1%, 16-0.1%, 32-0.0%, 64-0.0%, >-64-0.0% 1024.0 total. 1024.0 free. 0.0 used. 185554.7 avail Mem issued rwts: total=64728.0.0.0 short=0.0.0.0 dropped=0.0.0.0 latency : target=0, window=0, percentile=100.00%, depth=16 DTD IISED SHD S %CDII SMEM TIME+ COMMAND un status group 0 (all jobs): 387577 root 49.0 0:05.80 io wae worker-0 READ: bw=5396MiB/s (5658MB/s), 4096MiB/s-5396MiB/s (4295MB/s-5658MB/s), io=253GiB (271G 307549 root 0:21.39 io wae worker-0 307555 root 0.0 0:21.45 io wae worker-0 307567 root 0.0 .92 io wae worker-1 fio test: (g=0): rw=read. bs=(R) 4096KiB-4096KiB. (W) 4096KiB-4096KiB. (T) 4096KiB-4096KiB 307558 root fio-3.22 307556 root Starting 2 threads 307559 root 19.5 0:08.92 smbd obs: 2 (f=2): [R(2)][15.3%][r=6816MiB/s][r=1704 IOPS][eta 84m:14s] 307563 root 19.5 0:08.86 smbc 387557 root 0:09.11 smbd Task Manager 387568 root 0:09 38 smbd File Options View 307561 root 0:09.07 smbd 307534 root 0:09.00 smbd Processes Performance Users Details Services 307576 root 0:05.61 smbd 307562 root 0.1 0:08.93 smbd CPU Ethernet 307530 root 0:05.16 smbd 16% 2.78 GHz 307552 root 0.0 0:12.25 io wae worker-0 Throughput 417 root 0:03.58 kworker/0:2-event Memory 307183 root 0.3 0.0 0:00.61 kworker/u160:2-ml 12/512 GB (2%) 307568 root 0:00.02 kworker/29:0-ever 0.3 0.0 307588 root 0.0 Ethernet 1 root 0:02.84 systemd S: 17.4 Mbps R: 57.5 Gbps 0:00.13 kthreadd 2 root θ 0.0 3 root 0 -26 0:00.00 rcu qp Ethernet 4 root 0 -26 0.0 0:00.00 rcu par qp S: 32.0 Kbps R: 96.0 Kbps 6 root 0:00.00 kworker/0:0H-kblo 10 root 0 -26 0.0 0.0 0:00.00 mm percpu wg 11 root 0.0 0:00.32 ksoftirgd/0 12 root 0.0 0:03.17 rcu sched SLOT 4 Port Ethernet 13 root 0.0 0:00.03 migration/0 Connection type: 17.4 Mbps IPv4 address: 14 root 0.0 0.0 0:00.00 cpuhp/0 192.168.0.153 I Receive 15 root 0:00.00 cpuhp/1 IPv6 address: fe80::d5a5:8155:cccc:a4db%19 16 root 8.8 0.0 0:01.38 migration/1 57.5 Gbps 17 root 0:00.07 ksoftirad/1 19 root 8.8 0.0 0:00.00 kworker/1:0H-kblo 0 I 0:00.00 cpuhp/2 21 root 8.8 0.0 Rewer details Open Resource Monitor 22 root 0.0 0.0 0:01.37 migration/2 23 root 0:00.01 ksoftirad/2 5 items 25 root 0 -26 8.8 0.0 0:00.00 kworker/2:0H-kblos 26 root 8.8 0.0 0:00.00 cpuhp/3 н



0:01.39 migration/3

IORING_OP_SENDMSG prototyped (Part2)

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

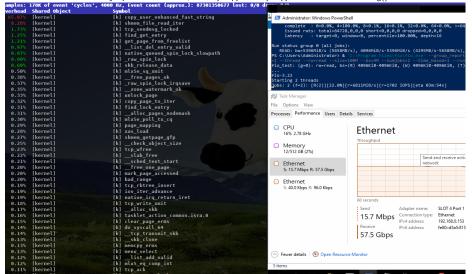




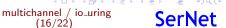


IORING_OP_SENDMSG prototyped (Part3)

The major problem still exists, memory copy done by copy_user_enhanced_fast_string()



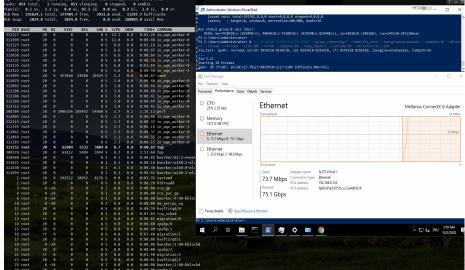




IORING_OP_SENDMSG/SPLICE prototyped (Part1)

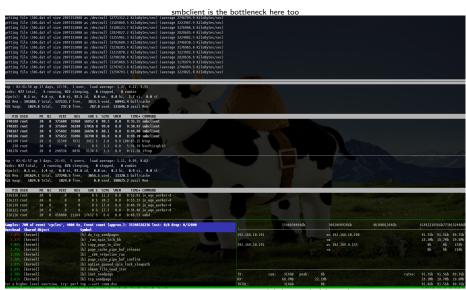
16 connections, "8.9 GBytes/s, smbd "5% cpu, (io_wqe_work 3%-12% cpu filesystem->pipe->socket), only "100% cpu in total.

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.





SerNet

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() 84:88:58 up 4 days, 23:82, 6 users, load average: 9.15, 3.56, 1.4 etting file \586.dat of size 2897152888 as /dev/null (2942528.3 KiloBytes/sec) (average 2943679.6 KiloBytes/sec) Tasks: 917 total, 14 running, 983 sleeping, 8 stopped, 8 zombie getting file \586.dat of size 2007152000 as /dev/null (2719787.2 KiloBytes/sec) (average 2841637.3 KiloBytes/sec) %Cpu(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 etting file \506.dat of size 2097152000 as /dev/null (3107738.5 KiloBytes/sec) (average 2958064.5 KiloBytes/sec) etting file \506.dat of size 2097152000 as /dev/null (2694736.5 KiloBytes/sec) (average 2714142.3 KiloBytes/sec) etting file \506.dat of size 2097152000 as /dev/null (2060334.8 KiloBytes/sec) (average 2733460.0 KiloBytes/sec) etting file \506.dat of size 2097152000 as /dev/null (3117198.9 KiloBytes/sec) (average 2090262.3 KiloBytes/sec) 322764 root etting file \586.dat of size 2897152888 as /dev/null (3847618.6 KiloRytes/sec) (average 2944358.1 KiloRytes/sec) etting file \506.dat of size 2097152000 as /dev/null (3098335.4 KiloBytes/sec) (average 2741473.6 KiloBytes/sec) 322768 root etting file \506.dat of size 2097152000 as /dev/null (2741632.8 KiloBytes/sec) (average 2840912.6 KiloBytes/sec) 322762 root etting file \50G.dat of size 2097152000 as /dev/null (3002932.1 KiloBytes/sec) (average 2830254.5 KiloBytes/sec) 322761 root etting file \506.dat of size 2097152000 as /dev/null (3126717.1 KiloBytes/sec) (average 2959135.8 KiloBytes/sec) 322766 root etting file \506.dat of size 2097152000 as /dev/null (3088939.0 KiloBytes/sec) (average 2091536.4 KiloBytes/sec) 322759 root etting file \586.dat of size 2097152000 as /dev/null (2515970.2 KiloBytes/sec) (average 2731748.8 KiloBytes/sec) 322782 root etting file \506.dat of size 2097152000 as /dev/null (2171791.9 KiloBytes/sec) (average 2709204.0 KiloBytes/sec) 322827 root setting file \586.dat of size 2897152888 as /dev/null (2921548.2 KiloRytes/sec) (2921ag 2944283.8 KiloRytes/sec) 322882 root petting file \586.dat of size 2897152888 as /dev/null (3893655.1 KiloRytes/sec) (2yerage 2743728.7 KiloRytes/sec) 322838 root getting file \586.dat of size 2897152888 as /dev/null (3893855.1 KiloRytes/sec) (average 2842525.3 KiloRytes/sec) 322772 root netting file \586.dat of size 2897152888 as /dev/null (3887341.7 KiloBytes/sec) (average 2881888.4 KiloBytes/sec) 322796 root getting file \586.dat of size 2897152888 as /dev/null (3187738.5 KiloBytes/sec) (average 2988879.4 KiloBytes/sec) 322888 root getting file \586.dat of size 2897152888 as /dev/null (3136293.6 KiloBytes/sec) (average 2893872.3 KiloBytes/sec) 322822 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 (3054336.9 KiloBytes/sec) (average 2945095.8 KiloBytes/sec) 318818 root etting file \586.dat of size 2897152888 as /dev/null (2745388.8 KiloBytes/sec) (average 2789462.2 KiloBytes/sec) 322833 root 28 etting file \506.dat of size 2097152000 as /dev/null (3117198.9 KiloBytes/sec) (average 2746070.8 KiloBytes/sec) 322854 root etting file \506.dat of size 2097152000 as /dev/null (3117198.9 KiloBytes/sec) (average 2844253.7 KiloBytes/sec) 322842 root 4.6 getting file \506.dat of size 2097152000 as /dev/null (2563203.7 KiloBytes/sec) (average 2078659.8 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 getting file \506.dat of size 2097152000 as /dev/null (2828728.9 KiloBytes/sec) (average 2732566.5 KiloBytes/sec) 318738 root getting file \506.dat of size 2097152000 as /dev/null (2771312.2 KiloBytes/sec) (average 2709897.3 KiloBytes/sec) 322836 root getting file \506.dat of size 2097152000 as /dev/null (3131498.0 KiloBytes/sec) (average 2846041.8 KiloBytes/sec) 322839 root 8:82.77 io wae worker-6 getting file \506.dat of size 2097152000 as /dev/null (3131498.0 KiloBytes/sec) (average 2748470.0 KiloBytes/sec) 322848 root getting file \506.dat of size 2097152000 as /dev/null (2595690.4 KiloBytes/sec) (average 2942472.7 KiloBytes/sec) 322865 root getting file \506.dat of size 2097152000 as /dev/null (3038575.2 KiloBytes/sec) (average 2957176.0 KiloBytes/sec) 322868 root getting file \506.dat of size 2097152000 as /dev/null (2976743.8 KiloBytes/sec) (average 2879300.8 KiloBytes/sec) 322887 root getting file \506.dat of size 2097152000 as /dev/null (3038575.2 KiloBytes/sec) (average 2095262.7 KiloBytes/sec) 3.6 0.0 0:02.33 io wge worker-8 getting file \586.dat of size 2097152000 as /dev/null (2024027.2 KiloBytes/sec) (average 2733199.6 KiloBytes/sec) 3.6 8.8 8:82.52 io wge worker-8 amples: 30M of event 'cycles', 1000 Hz, Event count (approx.): 526705589529 lost: 0/0 drop: 0/0 erhead Shared Object [k] copy user enhanced fast string [k] native queued spin lock slowpath [k] do top sendpages [k] raw spin lock bh [k] prb fill curr block.isra.0 [k] raw spin lock [k] copy page to iter [kernel] Ikl skh release data 2264268 neak: [k] check object size io_uring Stefan Metzmacher

More loopback testing on brand new hardware

- Recently I re-did the loopback read tests IORING_OP_SENDMSG/SPLICE (from /dev/shm/)
 - ▶ 1 connection, ~11 GBytes/s, smbd 7% cpu, with 4 io_wqe_work threads at 7%-50% cpu.
 - ▶ 4 connections, 24-30 GBytes/s, smbd 18% cpu, with 16 io_wqe_work threads at 3%-35% cpu.
- ► I also prototyped SMB2 writes with JORING_OP_RECVMSG/SPLICE (to /dev/null)
 - 1 connection, "7 GBytes/s and "2 pu, with 3 io wge work threads at %-20% cpu
 - 4 connections, ~10 GDV cs/h, smbd 15% cpu, with 12 io_wqe_wc/k threads at 1%-20% cpu.
- ▶ I tested with a Linux Kernel 5.10.25
 - In both cases the bottleneck is clearly on the smbclient side
 - We could post, similar changes to smbclient and add true multichanne 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, ~11 GBytes/s, smbd 7% cpu, with 4 io_wqe_work threads at 7%-50% cpu.
 - ▶ 4 connections, 24-30 GBytes/s, smbd 18% cpu, with 16 io_wqe_work threads at 3%-35% cpu.
- ► I also prototyped SMB2 writes with IORING_OP_RECVMSG/SPLICE (to /dev/null)
 - ▶ 1 connection, ~7 GBytes/s, smbd 5% cpu, with 3 io_wqe_work threads at 1%-20% cpu.
 - ▶ 4 connections, ~10 GBytes/s, smbd 15% cpu, with 12 io_wqe_work threads at 1%-20% cpu.
- ▶ I tested with a linux Kernel 5.10.25
 - In both cases the bottleneck is clearly on the smbclient side
 - We could be similar changes to smbclient and add true multichanne support
 - ▶ It seems that the filesystem->pipe->socket path is much better



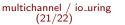
More loopback testing on brand new hardware

- Recently I re-did the loopback read tests IORING_OP_SENDMSG/SPLICE (from /dev/shm/)
 - ▶ 1 connection, ~11 GBytes/s, smbd 7% cpu, with 4 io_wqe_work threads at 7%-50% cpu.
 - ▶ 4 connections, 24-30 GBytes/s, smbd 18% cpu, with 16 io_wge_work threads at 3%-35% cpu.
- ► I also prototyped SMB2 writes with IORING_OP_RECVMSG/SPLICE (to /dev/null)
 - ▶ 1 connection, ~7 GBytes/s, smbd 5% cpu, with 3 io_wge_work threads at 1%-20% cpu.
 - 4 connections, ~10 GBytes/s, smbd 15% cpu, with 12 io_wge_work threads at 1%-20% cpu.
- ▶ I tested with a Linux Kernel 5.10.25
 - 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

- 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

 - Maybe it's possible to optimize this in future
- ► For SMB3 signing/encryption we may use:
 - IORING_OP_TEE with was blice could be used in order to still allow IORING_OP_SPLICE from no the filesystem
 - vmsplice may also need to be optimized and added to io-uring
 - With CBPF support in io-uring we might be able to offline signing encryption
- In the end SMB-Direct will also be able to reduce overhead
 - My smbdirect driver is still work in progress

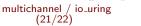






- recvmsg and splice deliver partial SMB packets to userspace
 - ▶ I tested with AF_KCM (Kernel Connection Multiplexor) and an eBPF helper
 - ▶ But MSG_WAITALL is the much simpler and faster solution
 - I also prototyped a SPLICE_F_WAITALL
 - ▶ eBPF support in io-uring would also be great for optimizations
- ▶ It also seems that socket->pipe->filesystem:
 - ▶ Does not implement zero copy for all cases
 - Maybe it's possible to optimize this in future
- For SMB3 signing/encryption we may use
 - IORING_OP_TEE with various could be used in order to still allow IORING_OP_SPLICE from to the filesystem
 - vmsplice may also need to be optimized and added to io-uring
 - With early support in io-uring we might be able to offline signing encryption
- In the end SMB-Direct will also be able to reduce overhead
 - My smbdirect driver is still work in progress.





- 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
- ► For SMB3 signing/encryption we may use:
 - ▶ IORING_OP_TEE with vmsplice could be used in order to still allow IORING_OP_SPLICE from/to the filesystem
 - vmsplice may also need to be optimized and added to io-uring
 - With eBPF support in io-uring we might be able to offline signing/encryption
- In the end SMB-Direct will also be able to reduce overhead
 - My smbdirect driver is still work in progress



- 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
- For SMB3 signing/encryption we may use:
 - ► IORING_OP_TEE with vmsplice could be used in order to still allow IORING_OP_SPLICE from/to the filesystem
 - vmsplice may also need to be optimized and added to io-uring
 - With eBPF support in io-uring we might be able to offline signing/encryption
- ▶ In the end SMB-Direct will also be able to reduce overhead
 - ▶ My smbdirect driver is still work in progress...







Questions? Feedback!

- Feedback regarding real world testing would be great!
- Stefan Metzmacher, metze@samba.org
- https://www.sernet.com
- https://samba.plus

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



