

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/>

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

What is SMB3 Multichannel?

- ▶ 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 ClientGUID
 - ▶ 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

What is SMB3 Multichannel?

- ▶ 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 ClientGUID
 - ▶ 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

What is SMB3 Multichannel?

- ▶ 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 ClientGUID
 - ▶ 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

What is SMB3 Multichannel?

- ▶ 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 ClientGUID
 - ▶ 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)
- ▶ 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)
- ▶ 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
- ▶ 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

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)

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

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

- ▶ 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

- ▶ 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.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.html>

Performance with MultiChannel, sendmsg()

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

```
top - 05:43:16 up 2 days, 44 min, 2 users, load average: 5.42, 3.22, 1.52
Threads: 823 total, 33 running, 790 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 6.3 sy, 0.0 ni, 93.4 id, 0.0 wa, 0.1 hi, 0.2 si, 0.0 st
MiB Mem : 191624.1 total, 182280.8 free, 2617.5 used, 6726.1 buff/cache
MiB Swap: 1024.0 total, 1024.0 free, 0.0 used, 185648.1 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	CPUI	%MEM	TIME	COMMAND
307312	root	20	0	2426196	63088	19104	R	06.8	0.0	0:52.24	send
307486	root	20	0	2426196	63408	19104	R	14.3	0.0	0:06.96	send
307412	root	20	0	2426196	65256	19104	R	14.8	0.0	0:06.92	send
307405	root	20	0	2426196	63144	19104	R	13.6	0.0	0:06.82	send
307410	root	20	0	2426196	64464	19104	R	13.6	0.0	0:06.77	send
307414	root	20	0	2426196	65520	19104	R	13.6	0.0	0:06.80	send
307422	root	20	0	2426196	68952	19104	R	13.6	0.0	0:06.78	send
307432	root	20	0	2426196	71592	19104	R	13.6	0.0	0:06.66	send
307400	root	20	0	2426196	63936	19104	R	13.3	0.0	0:06.50	send
307411	root	20	0	2426196	64992	19104	R	13.3	0.0	0:06.77	send
307413	root	20	0	2426196	65256	19104	R	13.3	0.0	0:06.68	send
307415	root	20	0	2426196	65256	19104	R	13.3	0.0	0:06.63	send
307410	root	20	0	2426196	66040	19104	R	13.3	0.0	0:06.69	send
307419	root	20	0	2426196	67104	19104	R	13.3	0.0	0:06.84	send
307420	root	20	0	2426196	67632	19104	R	13.3	0.0	0:06.76	send
307421	root	20	0	2426196	68160	19104	R	13.3	0.0	0:06.71	send
307423	root	20	0	2426196	69400	19104	R	13.3	0.0	0:06.68	send
307425	root	20	0	2426196	69400	19104	R	13.3	0.0	0:06.59	send
307420	root	20	0	2426196	70000	19104	R	13.3	0.0	0:06.59	send
307430	root	20	0	2426196	70000	19104	R	13.3	0.0	0:06.84	send
307433	root	20	0	2426196	72304	19104	R	13.3	0.0	0:06.61	send
307436	root	20	0	2426196	70000	19104	R	13.3	0.0	0:06.62	send
307429	root	20	0	2426196	70000	19104	R	13.0	0.0	0:06.07	send
307434	root	20	0	2426196	72304	19104	R	13.8	0.0	0:06.70	send
307435	root	20	0	2426196	72640	19104	R	13.0	0.0	0:06.71	send
307407	root	20	0	2426196	63672	19104	R	12.6	0.0	0:06.58	send
307416	root	20	0	2426196	66040	19104	R	12.6	0.0	0:06.68	send
307417	root	20	0	2426196	66312	19104	R	12.6	0.0	0:06.53	send
307427	root	20	0	2426196	70000	19104	R	12.6	0.0	0:06.87	send
307431	root	20	0	2426196	71064	19104	R	12.6	0.0	0:06.50	send
307424	root	20	0	2426196	69400	19104	R	12.3	0.0	0:06.65	send
307409	root	20	0	2426196	64200	19104	R	12.0	0.0	0:06.60	send
307404	root	20	0	2426196	62616	19104	D	11.3	0.0	0:01.61	send
307183	root	20	0	0	0	0	I	0.3	0.0	0:00.41	kwoker/u166:2-u1
307302	root	20	0	0	0	0	I	0.3	0.0	0:00.03	kwoker/z3:1-even
307452	root	20	0	62928	5536	3936	R	0.3	0.0	0:00.00	top
1	root	20	0	242512	10952	8176	S	0.0	0.0	0:02.84	system
2	root	20	0	0	0	0	S	0.0	0.0	0:00.13	khthread
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_par_gp
6	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kwoker/0:00-kblockd
10	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	mm_percpu_wq
11	root	20	0	0	0	0	S	0.0	0.0	0:00.32	ksoftirqd/0
12	root	20	0	0	0	0	I	0.0	0.0	0:00.17	rcu_sched
13	root	rt	0	0	0	0	S	0.0	0.0	0:00.03	migration/0
14	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/0
15	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/1
16	root	rt	0	0	0	0	S	0.0	0.0	0:01.38	migration/1

```
Administrator: Windows PowerShell
complete : 0.00%, 4=09.78, 0=0.38, 16=0.18, 32=0.08, 64=0.08, >=64=0.08
issued puts: total=4003,0,0,0 short=0,0,0,0 dropped=0,0,0,0
latency : target=0, window=0, percentile=100.00%, depth=16

Run status group 0 (all jobs):
  READ: bw=3260MiB/s (3425MB/s), 3260MiB/s-3260MiB/s (3425MB/s-3425MB/s), io=8000MiB (8395MB), run=2451-2451msrc
PS C:\Users\Administrator> . ".\Program Files\Fio\Fio.exe" --group_reporting=1 --name=fio_test --ioengine=windowsaio --iodepth=16 --direct
--io-read --io-read-ops=128 --io-write --io-write-ops=128 --time_based --runtime=60 --directory=C:\110
fio.test: (p=0) r=1,read,bs=(M) 4096KiB-4096KiB, (W) 4096KiB-4096KiB, (T) 4096KiB-4096KiB, ioengine=windowsaio, iodepth=16
...
FIO 3.22
Starting 2 threads
Jobs: 2 (*2): [R][T][117.3%][r=3812MiB/s][r=952.10PS][eta 04m:08s]
```

The screenshot shows the Windows Task Manager Performance tab. On the left, system metrics are listed: CPU (8% 2.78 GHz), Memory (12/312 GB (2%)), Ethernet (9.3 Mbps R; 31.9 Gbps), and Ethernet (40.0 Kbps R; 64.0 Kbps). The Ethernet section is expanded to show a throughput graph for the Mellanox ConnectX-6 Adaptor. The graph shows a peak throughput of 31.9 Gbps. Below the graph, network details are provided: Adapter name: SLOT 4 Port 1, Connection type: Ethernet, IPv4 address: 192.168.0.153, and IPv6 address: fe80:d5a5:8155:ccccca4b%19. At the bottom, there are links for 'Fewer details' and 'Open Resource Monitor'.

IOURING_OP_SENDMSG (Part1)

4 connections, ~6.8 GBytes/s, smbdc 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, 2.04, 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
Mem Mem : 191624.1 total, 182194.6 free, 2702.6 used, 6726.9 buff/cache
Mem Swap: 1024.0 total, 1024.0 free, 0.0 used, 185554.7 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
307577	root	20	0	0	0	0	R	49.0	0.0	0:05.00	io_wqe_worker-0
307549	root	20	0	0	0	0	S	46.0	0.0	0:21.39	io_wqe_worker-0
307555	root	20	0	0	0	0	R	44.0	0.0	0:21.45	io_wqe_worker-0
307567	root	20	0	0	0	0	S	29.8	0.0	0:09.92	io_wqe_worker-1
307558	root	20	0	663100	144024	18804	S	23.2	0.1	0:09.10	smbd
307556	root	20	0	663100	144024	18804	S	19.9	0.1	0:08.95	smbd
307559	root	20	0	663100	144024	18804	S	19.5	0.1	0:08.92	smbd
307563	root	20	0	663100	144024	18804	S	19.5	0.1	0:08.06	smbd
307557	root	20	0	663100	144024	18804	S	19.2	0.1	0:09.11	smbd
307560	root	20	0	663100	144024	18804	S	19.2	0.1	0:09.38	smbd
307561	root	20	0	663100	144024	18804	S	19.2	0.1	0:09.07	smbd
307534	root	20	0	663100	144024	18804	S	18.9	0.1	0:09.00	smbd
307576	root	20	0	663100	144024	18804	S	18.9	0.1	0:05.61	smbd
307562	root	20	0	663100	144024	18804	S	10.5	0.1	0:08.93	smbd
307530	root	20	0	663100	144024	18804	D	11.3	0.1	0:05.16	smbd
307552	root	20	0	0	0	0	S	9.3	0.0	0:12.25	io_wqe_worker-0
417	root	20	0	0	0	0	I	0.3	0.0	0:03.50	kworker/0:2-event
307183	root	20	0	0	0	0	I	0.3	0.0	0:00.61	kworker/u160:2-ml
307568	root	20	0	0	0	0	I	0.3	0.0	0:00.02	kworker/29:0-event
307588	root	20	0	62964	5532	3904	R	0.3	0.0	0:00.12	top
1	root	20	0	242512	10952	8176	S	0.0	0.0	0:02.04	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.13	kthreadd
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_par_gp
6	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/0:0H-kblou
10	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	mm_percpu_wq
11	root	20	0	0	0	0	S	0.0	0.0	0:00.32	ksftirqd/0
12	root	20	0	0	0	0	I	0.0	0.0	0:03.17	rcu_sched
13	root	rt	0	0	0	0	S	0.0	0.0	0:00.03	migration/0
14	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/0
15	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/1
16	root	rt	0	0	0	0	S	0.0	0.0	0:01.38	migration/1
17	root	20	0	0	0	0	S	0.0	0.0	0:00.07	ksftirqd/1
19	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/1:0H-kblou
21	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/2
22	root	rt	0	0	0	0	S	0.0	0.0	0:01.37	migration/2
23	root	20	0	0	0	0	S	0.0	0.0	0:00.01	ksftirqd/2
25	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/2:0H-kblou
26	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/3
27	root	rt	0	0	0	0	S	0.0	0.0	0:01.39	migration/3

The screenshot shows a Windows Task Manager window with the Performance tab selected. The system resources are:

- CPU: 16% 2.78 GHz
- Memory: 12/512 GB (2%)
- Ethernet: S: 17.4 Mbps R: 57.5 Gbps
- Ethernet: S: 32.0 Kbps R: 96.0 Kbps

The Ethernet section shows a throughput graph over 60 seconds. The Send rate is 17.4 Mbps and the Receive rate is 57.5 Gbps. The adapter name is SLOT 4 Port 1, the connection type is Ethernet, the IPv4 address is 192.168.0.153, and the IPv6 address is fe80:d5a5b155:cccc4db%19.

IOURING_OP_SENDMSG (Part3)

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

```
amples: 178K of event 'cycles', 4000 Hz, Event count (approx.): 87301350677 Lost: 0/0 d...
verhead Shared Object Symbol
65.07% [kernel] [k] copy_user_enhanced_fast_string
8.20% [kernel] [k] shmem_file_read_iter
1.73% [kernel] [k] tcp_sendmsg_locked
1.25% [kernel] [k] find_get_entry
1.21% [kernel] [k] get_page_from_freelist
0.97% [kernel] [k] __list_del_entry_valid
0.87% [kernel] [k] native_queued_spin_lock_slowpath
0.80% [kernel] [k] __raw_spin_lock
0.60% [kernel] [k] skb_release_data
0.50% [kernel] [k] mlx5e_sq_xmit
0.38% [kernel] [k] __free_pages_ok
0.37% [kernel] [k] __raw_spin_lock_irqsave
0.35% [kernel] [k] __zone_watermark_ok
0.33% [kernel] [k] unlock_page
0.32% [kernel] [k] copy_page_to_iter
0.31% [kernel] [k] find_lock_entry
0.31% [kernel] [k] __alloc_pages_nodemask
0.30% [kernel] [k] mlx5e_poll_tx_cq
0.29% [kernel] [k] page_mapping
0.28% [kernel] [k] xas_load
0.27% [kernel] [k] shmem_getpage_gfp
0.25% [kernel] [k] __check_object_size
0.23% [kernel] [k] tcp_wfree
0.22% [kernel] [k] __slab_free
0.21% [kernel] [k] __sched_text_start
0.20% [kernel] [k] __free_one_page
0.20% [kernel] [k] mark_page_accessed
0.20% [kernel] [k] bad_range
0.19% [kernel] [k] tcp_rbtrees_insert
0.19% [kernel] [k] iov_iter_advance
0.19% [kernel] [k] native_irq_return_iret
0.18% [kernel] [k] tcp_write_xmit
0.17% [kernel] [k] __alloc_skb
0.16% [kernel] [k] tasklet_action_common.isra.0
0.15% [kernel] [k] clear_page_erms
0.14% [kernel] [k] do_syscall_64
0.14% [kernel] [k] __tcp_transmit_skb
0.13% [kernel] [k] __skb_clone
0.13% [kernel] [k] memcpy_erms
0.13% [kernel] [k] menu_select
0.12% [kernel] [k] __list_add_valid
0.12% [kernel] [k] mlx5_eq_comp_int
0.11% [kernel] [k] tcp_ack
```

The screenshot shows the Windows Task Manager Performance tab. On the left, system metrics are listed: CPU (16% 2.78 GHz), Memory (12/512 GB (2%)), Ethernet (15.7 Mbps R; 57.5 Gbps), and another Ethernet interface (40.0 Kbps R; 96.0 Kbps). On the right, the 'Ethernet' section shows a throughput graph for 'Send and receive activity network' over 60 seconds. Below the graph, the following details are provided: Adapter name: SLOT 4 Port 1, Connection type: Ethernet, IPv4 address: 192.168.0.153, and IPv6 address: fe80::d5a5b15. The current throughput is shown as 15.7 Mbps for Send and 57.5 Gbps for Receive. At the bottom, there are links for 'Fewer details' and 'Open Resource Monitor'.

IOURING_OP_SENDMSG + IOURING_OP_SPLICE (Part1)

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

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

The screenshot displays a Windows system with a task manager window open. The task manager shows a high CPU usage of 25% (2.78 GHz) and a task named 'smbd' with a CPU usage of 3.0%. The Resource Monitor window is open, showing network performance for the Mellanox ConnectX-6 Adapter. The Ethernet adapter is selected, showing a throughput of 73.7 Mbps (Send) and 75.1 Gbps (Receive). The system is running Windows 10, and the task manager shows various system processes and services.

PID	USER	PR	NI	VIRT	RES	SHR	S	CPU	MEM	TIME	COMMAND
312117	root	20	0	0	0	0	0	5	12.3	0.0	0:01.26 io_wqeworker-0
311999	root	20	0	0	0	0	0	5	11.0	0.0	0:00.98 io_wqeworker-0
312125	root	20	0	0	0	0	0	5	8.6	0.0	0:01.19 io_wqeworker-0
312826	root	20	0	0	0	0	0	5	6.6	0.0	0:00.97 io_wqeworker-0
312836	root	20	0	0	0	0	0	5	6.6	0.0	0:00.94 io_wqeworker-0
312132	root	20	0	0	0	0	0	5	6.0	0.0	0:00.59 io_wqeworker-1
312135	root	20	0	0	0	0	0	5	6.0	0.0	0:01.04 io_wqeworker-0
312122	root	20	0	0	0	0	0	5	5.6	0.0	0:00.58 io_wqeworker-1
311994	root	20	0	457860	24880	18424	5	3.3	0.0	0:00.07 smbd	
312079	root	20	0	0	0	0	0	5	3.0	0.0	0:00.40 io_wqeworker-0
312892	root	20	0	0	0	0	0	5	3.0	0.0	0:00.44 io_wqeworker-0
312100	root	20	0	0	0	0	0	5	3.0	0.0	0:00.40 io_wqeworker-0
312106	root	20	0	0	0	0	0	5	3.0	0.0	0:00.41 io_wqeworker-0
312109	root	20	0	0	0	0	0	5	3.0	0.0	0:00.44 io_wqeworker-0
312112	root	20	0	0	0	0	0	5	3.0	0.0	0:00.41 io_wqeworker-0
308304	root	20	0	2986356	108452	54660	5	2.7	0.1	1:38.13 perf	
312895	root	20	0	0	0	0	0	5	2.7	0.0	0:00.46 io_wqeworker-0
312115	root	20	0	0	0	0	0	5	2.7	0.0	0:00.37 io_wqeworker-0
312145	root	20	0	0	0	0	0	5	2.7	0.0	0:00.18 io_wqeworker-1
312062	root	20	0	0	0	0	0	5	2.3	0.0	0:00.37 io_wqeworker-0
312869	root	20	0	0	0	0	0	5	2.3	0.0	0:00.35 io_wqeworker-0
312103	root	20	0	0	0	0	0	5	2.3	0.0	0:00.15 io_wqeworker-0
312151	root	20	0	62904	5532	3804	R	0.7	0.0	0:00.03 top	
308276	root	20	0	62812	5404	3844	5	0.3	0.0	3:57.64 top	
310569	root	20	0	0	0	0	I	0.3	0.0	0:00.02 kworker/61:2-event	
311821	root	20	0	0	0	0	I	0.3	0.0	0:00.10 kworker/u160:2-nl	
311830	root	20	0	0	0	0	I	0.3	0.0	0:00.30 kworker/u160:0-nl	
311894	root	20	0	0	0	0	I	0.3	0.0	0:00.42 kworker/u160:3-nl	
1	root	20	0	242512	18952	8176	5	0.0	0.0	0:03.35 systemd	
2	root	20	0	0	0	0	0	5	0.0	0:00.20 kthread	
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00 rcu_gp	
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00 rcu_par_gp	
6	root	0	-20	0	0	0	I	0.0	0.0	0:00.00 kworker/0:0H-kblockd	
10	root	0	-20	0	0	0	I	0.0	0.0	0:00.00 ms_percpu_wq	
11	root	20	0	0	0	0	0	5	0.0	0:00.39 ksoftirqd/0	
12	root	20	0	0	0	0	0	1	0.0	0:07.04 rcu_sched	
13	root	rt	0	0	0	0	0	5	0.0	0:00.05 migration/0	
14	root	20	0	0	0	0	0	5	0.0	0:00.00 cpuhp/0	
15	root	20	0	0	0	0	0	5	0.0	0:00.00 cpuhp/1	
16	root	rt	0	0	0	0	0	5	0.0	0:01.40 migration/1	
17	root	20	0	0	0	0	0	5	0.0	0:00.00 ksoftirqd/1	
19	root	0	-20	0	0	0	I	0.0	0.0	0:00.00 kworker/1:0H-kblockd	
21	root	20	0	0	0	0	0	5	0.0	0:00.00 cpuhp/2	
22	root	rt	0	0	0	0	0	5	0.0	0:01.40 migration/2	
23	root	20	0	0	0	0	0	5	0.0	0:00.01 ksoftirqd/2	
25	root	0	-20	0	0	0	I	0.0	0.0	0:00.00 kworker/2:0H-kblockd	

smbclient IORING_OP_SENDMSG/SPLICE (network)

4 connections, ~11 GBytes/s, smbld 8.6% cpu, with 4 io_wqe.work threads (pipe to socket) at ~20% cpu each.

smbclient is the bottleneck here too

```
getting file %S6.dat of size 2097152000 as /dev/null [2771312.2 KiBytes/sec] (average 2746784.9 KiBytes/sec)
getting file %S6.dat of size 2097152000 as /dev/null [3105609.5 KiBytes/sec] (average 3223967.9 KiBytes/sec)
getting file %S6.dat of size 2097152000 as /dev/null [3180123.7 KiBytes/sec] (average 3174986.8 KiBytes/sec)
getting file %S6.dat of size 2097152000 as /dev/null [2824827.2 KiBytes/sec] (average 2828665.4 KiBytes/sec)
getting file %S6.dat of size 2097152000 as /dev/null [3255961.3 KiBytes/sec] (average 3244082.5 KiBytes/sec)
getting file %S6.dat of size 2097152000 as /dev/null [2782688.3 KiBytes/sec] (average 2746838.3 KiBytes/sec)
getting file %S6.dat of size 2097152000 as /dev/null [3238283.4 KiBytes/sec] (average 3178965.8 KiBytes/sec)
getting file %S6.dat of size 2097152000 as /dev/null [3215878.2 KiBytes/sec] (average 3223992.8 KiBytes/sec)
getting file %S6.dat of size 2097152000 as /dev/null [2790190.4 KiBytes/sec] (average 2820636.8 KiBytes/sec)
getting file %S6.dat of size 2097152000 as /dev/null [3185689.5 KiBytes/sec] (average 3178974.8 KiBytes/sec)
getting file %S6.dat of size 2097152000 as /dev/null [2797813.8 KiBytes/sec] (average 2748894.5 KiBytes/sec)
getting file %S6.dat of size 2097152000 as /dev/null [3258783.1 KiBytes/sec] (average 3224021.8 KiBytes/sec)
```

```
top - 02:41:58 up 17 days, 17:34, 1 user, load average: 3.97, 4.22, 3.55
Tasks: 977 total, 5 running, 972 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.1 us, 4.0 sy, 0.0 ni, 93.5 id, 0.0 wa, 0.0 hi, 1.7 si, 0.0 st
Mem Mem : 131824.1 total, 127133.7 free, 3813.5 used, 60991.4 buff/cache
Mem Swap: 1824.0 total, 737.0 free, 287.0 used, 131646.0 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
740188	root	20	0	375608	35968	16852	R	99.3	0.0	0:35.55	smbclient
740185	root	20	0	375664	36180	17016	R	99.0	0.0	0:30.87	smbclient
740187	root	20	0	375652	35888	16896	R	88.1	0.0	0:44.08	smbclient
740186	root	20	0	375652	35896	16748	R	86.4	0.0	0:49.28	smbclient
100189	root	20	0	31548	7872	3412	S	2.0	0.0	100:05:15	top
238	root	20	0	0	0	0	S	1.3	0.0	5:56.30	ksftirqd/45
740176	root	20	0	249536	8076	5136	S	1.3	0.0	0:11.20	iftop

```
top - 02:41:57 up 3 days, 21:43, 5 users, load average: 1.11, 0.89, 0.62
Tasks: 977 total, 1 running, 876 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.1 us, 1.4 sy, 0.0 ni, 97.6 id, 0.0 wa, 0.1 hi, 0.9 si, 0.0 st
Mem Mem : 131824.1 total, 117240.5 free, 3955.5 used, 11338.1 buff/cache
Mem Swap: 1824.0 total, 1824.0 free, 0.0 used, 180675.2 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
316136	root	20	0	0	0	0	S	21.3	0.0	0:52.01	io_wqe_worker-0
316133	root	20	0	0	0	0	S	20.3	0.0	0:53.37	io_wqe_worker-0
316139	root	20	0	0	0	0	S	17.9	0.0	0:46.39	io_wqe_worker-0
316121	root	20	0	0	0	0	S	17.3	0.0	0:34.40	io_wqe_worker-0
316116	root	20	0	458880	21264	17652	S	6.6	0.0	0:46.53	smbd

Samples: 780 of event 'cycles', 4000 Hz, Event count (approx.): 35349326236 last: 0/0 drop: 0/32990

Overhead	Shared object	Symbol
7.85%	[kernel]	[k] do_tcp_sendpages
5.37%	[kernel]	[k] raw_spin_lock_bh
4.00%	[kernel]	[k] copy_page_to_iter
3.75%	[kernel]	[k] page_cache_pipe_buf_release
3.09%	[kernel]	[k] sbs_repoline_rsk
3.09%	[kernel]	[k] page_cache_pipe_buf_confirm
2.87%	[kernel]	[k] native_queued_spin_lock_slowpath
2.04%	[kernel]	[k] shmem_file_read_iter
2.04%	[kernel]	[k] inet_sendpage
2.03%	[kernel]	[k] tcp_sendpage

	1546038464gb	3892866928cb	4638891264cb	6184121056cb7738152448b
192.168.10.191	↔	↔	↔	↔
192.168.10.191	↔	↔	↔	↔
TX:	cus: 3146b	peak: 0b	rates: 91.7Gb	91.5Gb
RX:	68.7Mb	22.1Mb	18.3Mb	18.7Mb
TOTAL:	3146b	0b	91.8Gb	91.5Gb

for a higher level overview, try: perf top --sort comm,dsd

smbclient (R)G_OP_READMSG/SPICE (loopback)

8 connections, ~22 GBytes/s, smbdc 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()

getting file %SMB.dat of size 2097152000 as /dev/nvml	3075974.0 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	2945258.3 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	2717707.7 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	2951060.2 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	2203161.2 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	3107770.5 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	2694736.5 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	2680334.0 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	3117130.9 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	3047618.6 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	3088335.4 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	2741632.0 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	3002322.1 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	3126717.1 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	3080889.0 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	2515970.2 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	2127371.9 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	2923540.2 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	3083655.7 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	3093741.7 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	3107770.5 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	3136293.6 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	2752867.0 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	3064336.0 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	2745330.0 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	3117130.9 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	3117130.9 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	2596388.4 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	2530975.2 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	2530984.0 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	3083655.7 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	2828720.9 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	2771332.2 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	3133498.0 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	3133498.0 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	2596388.4 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	3003675.2 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	2927674.0 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	3083655.7 KiloBytes/sec
getting file %SMB.dat of size 2097152000 as /dev/nvml	2828427.2 KiloBytes/sec

```
top - 04:00:58 up 4 days, 23:02, 6 users, load average: 9.15, 3.56, 1.44
Tasks: 937 total, 14 running, 903 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.3 us, 11.2 sy, 0.0 ni, 06.1 id, 0.0 wa, 0.2 hi, 2.1 si, 0.0 st
MiB Mem: 191624.1 total, 170025.4 free, 3316.7 used, 11382.0 buff/cache
MiB Swap: 1824.0 total, 1824.0 free, 0.0 used, 108693.7 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	CU	MEM	TIME+	COMMAND
322763	root	20	0	376228	36620	17364	R	02.5	0.0	1:26.20	smbclient
322764	root	20	0	360836	28192	17120	R	01.5	0.0	1:26.18	smbclient
322765	root	20	0	360040	28516	17164	R	01.0	0.0	1:25.16	smbclient
322766	root	20	0	376244	36740	17468	R	79.0	0.0	1:25.73	smbclient
322767	root	20	0	376236	36600	17228	R	79.0	0.0	1:24.42	smbclient
322768	root	20	0	360040	28540	17464	R	79.5	0.0	1:25.83	smbclient
322759	root	20	0	376140	36494	17312	R	70.1	0.0	1:24.31	smbclient
322782	root	20	0	360040	0	0	S	23.0	0.0	0:14.04	io_wqe_worker-0
322827	root	20	0	0	0	0	S	23.5	0.0	0:12.77	io_wqe_worker-0
322882	root	20	0	0	0	0	S	22.8	0.0	0:14.36	io_wqe_worker-0
322838	root	20	0	0	0	0	S	22.8	0.0	0:12.96	io_wqe_worker-0
322772	root	20	0	458260	21480	17596	R	22.5	0.0	0:22.45	smbd
322796	root	20	0	0	0	0	S	22.2	0.0	0:11.00	io_wqe_worker-0
322880	root	20	0	0	0	0	S	22.5	0.0	0:14.13	io_wqe_worker-0
322822	root	20	0	0	0	0	R	21.5	0.0	0:12.86	io_wqe_worker-0
322810	root	20	0	0	0	0	S	19.2	0.0	0:12.71	io_wqe_worker-0
310810	root	20	0	244876	6976	4900	S	9.3	0.0	1:31.29	iftop
322833	root	20	0	0	0	0	S	5.3	0.0	0:02.58	io_wqe_worker-0
322854	root	20	0	0	0	0	S	5.0	0.0	0:02.58	io_wqe_worker-0
322842	root	20	0	0	0	0	S	4.6	0.0	0:02.70	io_wqe_worker-0
322861	root	20	0	0	0	0	S	4.6	0.0	0:02.69	io_wqe_worker-0
322880	root	20	0	0	0	0	S	4.6	0.0	0:02.54	io_wqe_worker-0
322862	root	20	0	0	0	0	S	4.6	0.0	0:02.78	io_wqe_worker-0
310730	root	20	0	3837184	172756	54344	S	4.3	0.1	1:49.89	perf
322836	root	20	0	0	0	0	S	4.3	0.0	0:02.61	io_wqe_worker-0
322839	root	20	0	0	0	0	S	4.3	0.0	0:02.77	io_wqe_worker-0
322846	root	20	0	0	0	0	S	4.0	0.0	0:02.52	io_wqe_worker-0
322865	root	20	0	0	0	0	S	4.0	0.0	0:02.68	io_wqe_worker-0
322860	root	20	0	0	0	0	S	4.0	0.0	0:02.66	io_wqe_worker-0
322887	root	20	0	0	0	0	S	4.0	0.0	0:02.57	io_wqe_worker-0
322845	root	20	0	0	0	0	S	3.6	0.0	0:02.58	io_wqe_worker-0
322856	root	20	0	0	0	0	S	3.6	0.0	0:02.33	io_wqe_worker-0
322858	root	20	0	0	0	0	S	3.6	0.0	0:02.52	io_wqe_worker-0

```
Samples: 30k of event 'cycles', 1000 Hz, Event count (approx.): 52678559529 last: 0/0 drop: 0/0
Overhead Shared object (Symbol)
 6.01% [kernel] [k] copy_user_enhanced_fast_string
 6.00% [kernel] [k] native_queued_spin_lock_slowpath
 5.90% [kernel] [k] tcpack_rev
 3.79% [kernel] [k] do_tcp_sendpages
 3.20% [kernel] [k] raw_spin_lock_bh
 3.21% [kernel] [k] prb_fill_curr_block.isra.0
 3.03% [kernel] [k] raw_spin_lock
 0.92% [kernel] [k] copy_page_to_iter
 0.89% [kernel] [k] skb_release_data
 0.89% [kernel] [k] __check_object_size
For a higher level overview, try: perf top --sort comm,dsu
```

```
1575379286b 3151870406b 4726614016b 6382151680b877689346b
127.0.0.1 <= 127.0.0.1 181b 181b 1806b
ob ob ob
Tx: cum: 226476b peak: 6.596b rates: 181b 181b 1806b
Rx: ob ob ob
TOTAL: 226476b 6.596b 181b 181b 1806b
```

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.
- ▶ 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 bottleneck 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 bottleneck 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 bottleneck 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->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`

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, metze@samba.org
- ▶ <https://www.sernet.com>
- ▶ <https://samba.plus>

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