

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/>

- ▶ 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

- ▶ 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)

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 "[Ringling 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)

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/RECWRITE
 - ▶ 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>

SDC21 SAMBA+

Stefan Metzmacher

multichannel / io_uring
(11/22)

SerNet

Performance with MultiChannel, sendmsg()

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

```
top - 01:51:16 up 2 days, 44 min, 1 user, load average: 5.62, 3.22, 1.53
Threads: 823 total, 33 running, 790 sleeping, 0 stopped, 0 zombie
%mem: 0.8 free, 0.2 buff, 0.8 mem, 95.4 free, 0.8 swp, 0.1, 0.2, 0.1, 0.4 in
MiB Mem: 102462.1 total, 182288.4 free, 2817.5 used, 8228.1 buff/cache
MiB Swap: 1828.0 total, 1828.0 free, 0.0 used, 135849.1 avail Mem
```

PID	USER	PR	NI	VSZ	RSS	VSZ %	MEM %	MPUR	TIME	COMMAND
387272	root	0	0	2426126	60888	10184	0.6	0.0	0:12.24	sshd
387466	root	0	0	2426126	65968	10184	0.7	0.0	0:16.36	sshd
387412	root	0	0	2426126	65256	10184	0.6	0.0	0:16.32	sshd
387496	root	0	0	2426126	61344	10184	0.5	0.0	0:16.32	sshd
387418	root	0	0	2426126	64648	10184	0.5	0.0	0:16.77	sshd
387416	root	0	0	2426126	61528	10184	0.5	0.0	0:16.80	sshd
387422	root	0	0	2426126	60952	10184	0.5	0.0	0:16.78	sshd
387432	root	0	0	2426126	71582	10184	0.7	0.0	0:16.66	sshd
387468	root	0	0	2426126	61876	10184	0.5	0.0	0:16.58	sshd
387411	root	0	0	2426126	64892	10184	0.5	0.0	0:16.77	sshd
387413	root	0	0	2426126	65256	10184	0.5	0.0	0:16.69	sshd
387415	root	0	0	2426126	65028	10184	0.5	0.0	0:16.53	sshd
387417	root	0	0	2426126	64848	10184	0.5	0.0	0:16.49	sshd
387419	root	0	0	2426126	61344	10184	0.5	0.0	0:16.98	sshd
387420	root	0	0	2426126	61632	10184	0.5	0.0	0:16.76	sshd
387421	root	0	0	2426126	61168	10184	0.5	0.0	0:16.71	sshd
387423	root	0	0	2426126	64648	10184	0.5	0.0	0:16.59	sshd
387425	root	0	0	2426126	69488	10184	0.7	0.0	0:16.59	sshd
387426	root	0	0	2426126	70888	10184	0.7	0.0	0:16.59	sshd
387427	root	0	0	2426126	72184	10184	0.7	0.0	0:16.58	sshd
387428	root	0	0	2426126	72284	10184	0.7	0.0	0:16.78	sshd
387429	root	0	0	2426126	70888	10184	0.6	0.0	0:16.47	sshd
387430	root	0	0	2426126	72284	10184	0.7	0.0	0:16.78	sshd
387431	root	0	0	2426126	70448	10184	0.6	0.0	0:16.71	sshd
387432	root	0	0	2426126	70888	10184	0.6	0.0	0:16.58	sshd
387433	root	0	0	2426126	72184	10184	0.7	0.0	0:16.58	sshd
387434	root	0	0	2426126	70888	10184	0.6	0.0	0:16.58	sshd
387435	root	0	0	2426126	70448	10184	0.6	0.0	0:16.58	sshd
387436	root	0	0	2426126	68632	10184	0.5	0.0	0:16.53	sshd
387437	root	0	0	2426126	70888	10184	0.6	0.0	0:16.67	sshd
387438	root	0	0	2426126	71884	10184	0.7	0.0	0:16.58	sshd
387439	root	0	0	2426126	69488	10184	0.5	0.0	0:16.65	sshd
387440	root	0	0	2426126	65028	10184	0.5	0.0	0:16.68	sshd
387441	root	0	0	2426126	65028	10184	0.5	0.0	0:16.68	sshd
387442	root	0	0	2426126	62116	10184	0.5	0.0	0:16.61	sshd
387283	root	0	0	0	0	0	0.1	0.5	0:00.01	ksmthread(18173-46)
387282	root	0	0	0	0	0	0.1	0.5	0:00.03	ksmthread(21117000)
387452	root	0	0	812928	1576	3038	0.0	0.0	0:16.80	top
3	root	0	0	974212	38912	8176	0.9	0.0	0:16.80	top
2	root	0	0	0	0	0	0.0	0.0	0:04.11	ls1/smbd
3	root	0	-20	0	0	0	0.1	0.0	0:04.00	rcv_xp
4	root	0	-20	0	0	0	0.1	0.0	0:04.00	rcv_xp
5	root	0	-20	0	0	0	0.1	0.0	0:04.00	rcv_xp
6	root	0	-20	0	0	0	0.1	0.0	0:04.00	rcv_xp
7	root	0	-20	0	0	0	0.1	0.0	0:04.00	rcv_xp
8	root	0	-20	0	0	0	0.1	0.0	0:04.00	rcv_xp
9	root	0	-20	0	0	0	0.1	0.0	0:04.00	rcv_xp
10	root	0	-20	0	0	0	0.1	0.0	0:04.00	rcv_xp
11	root	0	-20	0	0	0	0.1	0.0	0:04.00	rcv_xp
12	root	0	-20	0	0	0	0.1	0.0	0:04.00	rcv_xp
13	root	0	-20	0	0	0	0.1	0.0	0:04.00	rcv_xp
14	root	0	-20	0	0	0	0.1	0.0	0:04.00	rcv_xp
15	root	0	-20	0	0	0	0.1	0.0	0:04.00	rcv_xp
16	root	0	-20	0	0	0	0.1	0.0	0:04.00	rcv_xp

The screenshot shows the Windows Task Manager Performance tab for the Ethernet adapter. The network usage is 319 Gbps, which is 93.3% of the 340 Gbps capacity. The adapter is identified as Mellanox ConnectX-6 Adapter. The interface is Gigabit Ethernet. The screenshot also shows the Windows System Information window in the background, displaying system statistics such as CPU usage (0.2%), memory usage (0.1%), and disk usage (0.1%).

SDC21 SAMBA+

Stefan Metzmacher

multichannel / io_uring
(12/22)

SerNet

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

SDC²¹

SAMBA⁺

Stefan Metzmacher

multichannel / io_uring
(19/22)

SerNet

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

SDC²¹

SAMBA⁺

Stefan Metzmacher

multichannel / io_uring
(20/22)

SerNet

Future Improvements

- ▶ recvmmsg 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>