

io_uring

Status Update within Samba

Stefan Metzmacher <metze@samba.org>

Samba Team / SerNet

2023-09-20

<https://samba.org/~metze/presentations/2023/SDC/>

- ▶ What is io-uring?
- ▶ io-uring for Samba
- ▶ Performance research, prototyping and ideas
- ▶ The road to upstream
- ▶ Future Improvements
- ▶ Questions? Feedback!

- ▶ 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 storage developer conference 2021:
 - ▶ See <https://samba.org/~metze/presentations/2021/SDC/>
 - ▶ It explains the milestones and updates up to Samba 4.15 (in detail)
- ▶ I gave a similar talk at the SambaXP conference 2023:
 - ▶ See <https://samba.org/~metze/presentations/2023/SambaXP/>
 - ▶ It explains the milestones and updates up to Samba 4.19 (in detail)

- ▶ 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 storage developer conference 2021:
 - ▶ See <https://samba.org/~metze/presentations/2021/SDC/>
 - ▶ It explains the milestones and updates up to Samba 4.15 (in detail)
- ▶ I gave a similar talk at the SambaXP conference 2023:
 - ▶ See <https://samba.org/~metze/presentations/2023/SambaXP/>
 - ▶ It explains the milestones and updates up to Samba 4.19 (in detail)

- ▶ 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 storage developer conference 2021:
 - ▶ See <https://samba.org/~metze/presentations/2021/SDC/>
 - ▶ It explains the milestones and updates up to Samba 4.15 (in detail)
- ▶ I gave a similar talk at the SambaXP conference 2023:
 - ▶ See <https://samba.org/~metze/presentations/2023/SambaXP/>
 - ▶ It explains the milestones and updates up to Samba 4.19 (in detail)

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 event 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 event 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



- ▶ Between userspace and filesystem (available from 5.1):
 - ▶ IORING_OP_READV, IORING_OP_WRITEV and IORING_OP_FSYNC
 - ▶ Supports buffered and direct io
 - ▶ IORING_OP_FSETXATTR, IORING_OP_FGETXATTR (from 5.19)
 - ▶ IORING_OP_GETDENTS, under discussion, but seems to be tricky
 - ▶ IORING_OP_FADVISE (from 5.6)
- ▶ 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_OP_SETXATTR, IORING_OP_GETXATTR (from 5.19)



- ▶ Between userspace and filesystem (available from 5.1):
 - ▶ IORING_OP_READV, IORING_OP_WRITEV and IORING_OP_FSYNC
 - ▶ Supports buffered and direct io
 - ▶ IORING_OP_FSETXATTR, IORING_OP_FGETXATTR (from 5.19)
 - ▶ IORING_OP_GETDENTS, under discussion, but seems to be tricky
 - ▶ IORING_OP_FADVISE (from 5.6)
- ▶ 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_OP_SETXATTR, IORING_OP_GETXATTR (from 5.19)



- ▶ 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)
 - ▶ Maybe using IOSQE_ASYNC in order to avoid inline memcpy
 - ▶ IORING_OP_SPLICE, IORING_OP_TEE
 - ▶ IORING_OP_SENDMSG_ZC, zero copy with an extra completion (from 6.1)
 - ▶ IORING_OP_GET_BUF, under discussion to replace IORING_OP_SPLICE



- ▶ 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



- ▶ 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>



- ▶ 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>



- ▶ 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>

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, 2.84, 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
MiB Mem : 191624.1 total, 182194.6 free, 2702.6 used, 6726.9 buff/cache
MiB 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	0:05.80	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	29.0	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	0:08.86	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	18.5	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	0:03.58	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/20:0-event
307580	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	0:02.84	system
2	root	20	0	0	0	0	S	0.0	0.0	0:00.13	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-kbLo
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	kssoftirqd/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	kssoftirqd/1
19	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/1:0H-kbLo
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	kssoftirqd/2
25	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/2:0H-kbLo
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

Administrator: Windows PowerShell

```
complete : 0=0.0%, 4=100.0%, 0=0.1%, 16=0.1%, 32=0.0%, 64=0.0%, >=64=0.0%
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
```

Run status group 0 (all jobs):
READ: bw=5396MiB/s (5658MB/s), 4096KiB/s-5396MiB/s (4295MB/s-5658MB/s), io=253GiB (2716
PS C:\Users\Administrator> C:\Program Files\Fio\Fio.exe --group_reporting=1 --name=fio
C: --thread --rw-read --size=100M --bs=4K --numjobs=2 --time_based=1 --runtime=90 --direct
fio_test: (g=0): rw=read, bs=(R) 4096KiB-4096KiB, (W) 4096KiB-4096KiB, (T) 4096KiB-4096KiB
...
Fio-3.22
Starting 2 threads
Jobs: 2 (F+2): [R(2)][15.3%][r=6816MiB/s][r=1704 IOPS][eta 04m:14s]

Task Manager Performance View

- 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

Ethernet Throughput

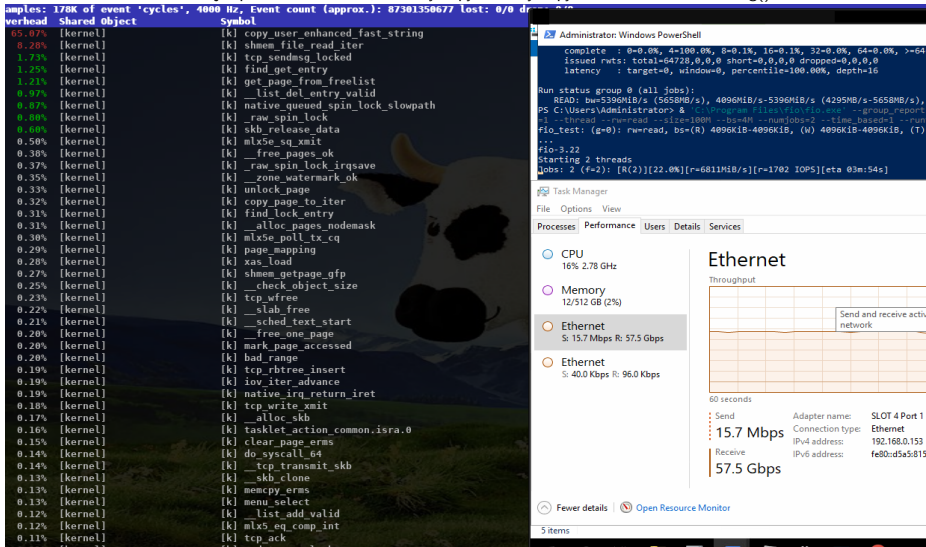
60 seconds

- Send: 17.4 Mbps
- Receive: 57.5 Gbps

Adapter name: SLOT 4 Port 1
Connection type: Ethernet
IPv4 address: 192.168.0.153
IPv6 address: fe80:d5a5:8155:ccccca4db%19

IOURING_OP_SENDMSG (Part2)

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



The screenshot displays the Windows Task Manager Performance tab. On the left, a list of system metrics is shown:

- CPU: 16% 2.78 GHz
- Memory: 12/512 GB (2%)
- Ethernet: S: 15.7 Mbps R: 57.5 Gbps
- Ethernet: S: 40.0 Kbps R: 96.0 Kbps

The Ethernet section is expanded to show detailed statistics:

- Send: 15.7 Mbps
- Receive: 57.5 Gbps
- Adapter name: SLOT 4 Port 1
- Connection type: Ethernet
- IPv4 address: 192.168.0.153
- IPv6 address: fe80::d5a5b15

At the top right, a PowerShell window shows the output of the `Get-Process` command, displaying details for the `System Idle Process` and `System` processes, including CPU usage, memory usage, and I/O statistics.

The background of the Task Manager window features a cow, which is a common meme image.

IORING_OP_SENDMSG + IORING_OP_SPLICE (Part 1)

16 connections, ~8.9 GBytes/s, smbdc ~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"

```
top - 04:59:15 up 3 days, 0 min, 4 users, load average: 0.63, 0.54, 0.26
tasks: 854 total, 1 running, 853 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.1 us, 1.2 sy, 0.0 ni, 97.1 id, 0.0 wa, 0.2 hi, 1.4 si, 0.0 st
Mem Mem: 191624.1 total, 177404.7 free, 2931.6 used, 112.7 b, 1.4 si, 0.0 st
Mem Swap: 1024.0 total, 1024.0 free, 0.0 used, 180093.9 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
312117	root	0	0	0	0	0	5	12.3	0.0	0:01.26	io_wqe_worker-0
311999	root	20	0	0	0	0	5	11.0	0.0	0:00.19	io_wqe_worker-0
312125	root	20	0	0	0	0	5	8.6	0.0	0:01.19	io_wqe_worker-0
312126	root	20	0	0	0	0	5	6.6	0.0	0:01.19	io_wqe_worker-0
312130	root	20	0	0	0	0	5	6.6	0.0	0:00.94	io_wqe_worker-0
312132	root	20	0	0	0	0	5	6.8	0.0	0:00.59	io_wqe_worker-1
312135	root	20	0	0	0	0	5	6.8	0.0	0:01.04	io_wqe_worker-0
312122	root	20	0	0	0	0	5	5.6	0.0	0:00.58	io_wqe_worker-1
311994	root	20	0	457060	24880	18424	S	5.3	0.0	0:00.07	smbd
312079	root	20	0	0	0	0	5	3.0	0.0	0:00.40	io_wqe_worker-0
312092	root	20	0	0	0	0	5	3.0	0.0	0:00.44	io_wqe_worker-0
312100	root	20	0	0	0	0	5	3.0	0.0	0:00.40	io_wqe_worker-0
312106	root	20	0	0	0	0	5	3.0	0.0	0:00.41	io_wqe_worker-0
312109	root	20	0	0	0	0	5	3.0	0.0	0:00.44	io_wqe_worker-0
312112	root	20	0	0	0	0	5	2.0	0.0	0:00.41	io_wqe_worker-0
308304	root	20	0	2996356	108452	54660	S	2.7	0.1	1:38.13	perf
312095	root	20	0	0	0	0	5	2.7	0.0	0:00.46	io_wqe_worker-0
312115	root	20	0	0	0	0	5	2.7	0.0	0:00.37	io_wqe_worker-0
312145	root	20	0	0	0	0	5	2.7	0.0	0:00.18	io_wqe_worker-1
312062	root	20	0	0	0	0	5	2.3	0.0	0:00.37	io_wqe_worker-0
312069	root	20	0	0	0	0	5	2.3	0.0	0:00.35	io_wqe_worker-0
312103	root	20	0	0	0	0	5	2.3	0.0	0:00.15	io_wqe_worker-0
312151	root	20	0	62904	5532	3084	R	0.7	0.0	0:00.03	top
30076	root	20	0	62812	5404	3044	S	0.3	0.0	3:57.04	top
31050	root	20	0	0	0	0	1	0.3	0.0	0:00.02	ksworker/61:2-avnet
311821	root	20	0	0	0	0	1	0.3	0.0	0:00.18	ksworker/u168:2-nl
311830	root	20	0	0	0	0	1	0.3	0.0	0:00.30	ksworker/u168:0-nl
311894	root	20	0	0	0	0	1	0.3	0.0	0:00.42	ksworker/u168:3-nl
1	root	20	0	242512	10952	8176	S	0.0	0.0	0:03.35	systemd
2	root	20	0	0	0	0	5	0.0	0.0	0:00.20	ktreaddd
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	ksworker/0:0H-kblock
10	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	me_percpu_wq
11	root	20	0	0	0	0	5	0.0	0.0	0:00.30	kssoftirqd/0
12	root	20	0	0	0	0	1	0.0	0.0	0:07.04	rcu_sched
13	root	rt	0	0	0	0	5	0.0	0.0	0:00.05	migration/0
14	root	20	0	0	0	0	5	0.0	0.0	0:00.00	cpulp/0
15	root	20	0	0	0	0	5	0.0	0.0	0:00.00	cpulp/1
16	root	rt	0	0	0	0	5	0.0	0.0	0:01.40	migration/1
17	root	20	0	0	0	0	5	0.0	0.0	0:00.00	kssoftirqd/1
19	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	ksworker/1:0H-kblock
21	root	20	0	0	0	0	5	0.0	0.0	0:00.00	cpulp/2
22	root	rt	0	0	0	0	5	0.0	0.0	0:01.40	migration/2
23	root	20	0	0	0	0	5	0.0	0.0	0:00.01	kssoftirqd/2
25	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	ksworker/2:0H-kblock
26	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	ksworker/2

The screenshot shows the Windows Task Manager Performance tab. The 'Ethernet' section is highlighted, showing a throughput of 73.7 Mbps (Send) and 75.1 Gbps (Receive). The adapter is identified as 'Mellanox ConnectX-6 Adapter'. The IP address is 192.168.0.153. The task manager also shows other system metrics like CPU (25% at 2.78 GHz) and Memory (15/512 GB at 3%).

smbclient IORING_OP_SENDMSG/SPLICE (network)

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

smbclient is the bottleneck here too

```
getting file %506.dat of size 2097152000 as /dev/null [2771312.2 KiloBytes/sec] (average 2746784.9 KiloBytes/sec)
getting file %506.dat of size 2097152000 as /dev/null [3185609.5 KiloBytes/sec] (average 3223967.9 KiloBytes/sec)
getting file %506.dat of size 2097152000 as /dev/null [3180233.7 KiloBytes/sec] (average 3174806.8 KiloBytes/sec)
getting file %506.dat of size 2097152000 as /dev/null [2824827.2 KiloBytes/sec] (average 2822665.4 KiloBytes/sec)
getting file %506.dat of size 2097152000 as /dev/null [3255961.3 KiloBytes/sec] (average 3244082.5 KiloBytes/sec)
getting file %506.dat of size 2097152000 as /dev/null [2782680.3 KiloBytes/sec] (average 2746830.3 KiloBytes/sec)
getting file %506.dat of size 2097152000 as /dev/null [3238283.2 KiloBytes/sec] (average 3178965.8 KiloBytes/sec)
getting file %506.dat of size 2097152000 as /dev/null [3215870.2 KiloBytes/sec] (average 3223992.8 KiloBytes/sec)
getting file %506.dat of size 2097152000 as /dev/null [2790190.4 KiloBytes/sec] (average 2822663.6 KiloBytes/sec)
getting file %506.dat of size 2097152000 as /dev/null [3185609.5 KiloBytes/sec] (average 3178974.8 KiloBytes/sec)
getting file %506.dat of size 2097152000 as /dev/null [2787813.8 KiloBytes/sec] (average 2748804.5 KiloBytes/sec)
getting file %506.dat of size 2097152000 as /dev/null [3258783.1 KiloBytes/sec] (average 3244021.8 KiloBytes/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, 4.6 sy, 0.0 ni, 93.5 id, 0.0 wa, 0.0 hi, 1.7 si, 0.0 st
Mem: 191888.7 total, 127137.0 free, 3433.5 used, 60941.4 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	35960	16852	R	99.3	0.0	0:35.55	smbclient
740185	root	20	0	375664	36180	17016	R	99.0	0.0	9:30.87	smbclient
740187	root	20	0	375692	35888	16996	R	88.1	0.0	0:44.88	smbclient
740186	root	20	0	375652	35896	16748	R	85.4	0.0	0:49.28	smbclient
180180	root	20	0	33540	7872	3412	S	7.0	0.0	100:03.15	htop
238	root	20	0	0	0	0	S	1.3	0.0	5:56.39	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: 877 total, 1 running, 876 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.1 us, 1.4 sy, 0.0 ni, 97.8 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
Mem: 191824.1 total, 137240.5 free, 3995.5 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_wqeworker-0
316133	root	20	0	0	0	0	S	20.3	0.0	0:53.17	io_wqeworker-0
316139	root	20	0	0	0	0	S	17.9	0.0	0:40.39	io_wqeworker-0
316121	root	20	0	0	0	0	S	17.3	0.0	0:34.48	io_wqeworker-0
316116	root	20	0	458080	21264	17652	S	8.6	0.0	0:46.53	smbd

```
Samples: 780 of event 'cycles', 4000 Hz, Event count (approx.): 35348326236 last: 0/0 drop: 0/32800
```

Overhead	Shared object	Symbol
7.85%	[kernel]	[k] do_tcp_sendpages
5.37%	[kernel]	[k] raw_spin_lock_bh
4.06%	[kernel]	[k] copy_page_to_iter
3.75%	[kernel]	[k] page_cache_pipe_buf_release
3.09%	[kernel]	[k] x86_retpollin_rax
3.09%	[kernel]	[k] page_cache_pipe_buf_confirm
2.87%	[kernel]	[k] native_queued_spin_lock_slowpath
2.84%	[kernel]	[k] shaem_file_read_iter
2.78%	[kernel]	[k] inet_sendpage
2.63%	[kernel]	[k] tcp_sendpage

For a higher level overview, try: perf top --sort comm,dso

	1546838464cb	3892866928cb	4638801264cb	6184121856cb7738152448cb
192.168.10.191	⇒	192.168.10.190		91.7Gb 91.5Gb 89.7Gb
192.168.10.191	⇐		192.168.0.153	18.3Gb 18.7Gb 18.6Gb
			⇐	0b 0b 238b
				0b 0b 210b
TX:	cur: 3146B	peak: 0b		rates: 91.7Gb 91.5Gb 89.7Gb
RX:	68.7MB	22.1Mb		18.3Gb 18.7Gb 18.6Gb
TOTAL:	3146B	0b		91.8Gb 91.5Gb 89.7Gb



Stefan Metzmacher

io_uring (13/21)



smbclient (R)GOP(ED)MSG/SPLICE(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 %S0.dat of size 2097152000 as /dev/null (3075974.0 KiBytes/sec) (average 2688080.8 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (2942520.3 KiBytes/sec) (average 2943670.6 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (2710770.7 KiBytes/sec) (average 2841637.3 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (2951800.2 KiBytes/sec) (average 2979437.6 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (2301612.2 KiBytes/sec) (average 2739378.9 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (3107770.5 KiBytes/sec) (average 2958066.5 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (2694736.5 KiBytes/sec) (average 2714142.3 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (2860334.3 KiBytes/sec) (average 2733560.8 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (3117180.9 KiBytes/sec) (average 2898262.3 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (3047610.6 KiBytes/sec) (average 2944350.1 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (3088355.4 KiBytes/sec) (average 2741473.6 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (2741632.0 KiBytes/sec) (average 2449162.0 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (3002932.1 KiBytes/sec) (average 2888254.5 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (3126712.1 KiBytes/sec) (average 2951935.8 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (3080890.9 KiBytes/sec) (average 2891536.4 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (2515970.2 KiBytes/sec) (average 2731740.8 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (2127371.9 KiBytes/sec) (average 2709740.8 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (2923540.2 KiBytes/sec) (average 2844283.8 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (3083655.5 KiBytes/sec) (average 2743720.7 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (3093655.5 KiBytes/sec) (average 2842526.3 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (3007341.7 KiBytes/sec) (average 2818004.4 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (3107770.5 KiBytes/sec) (average 2960079.0 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (3136293.6 KiBytes/sec) (average 2939302.3 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (2752687.6 KiBytes/sec) (average 2731050.3 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (3084336.9 KiBytes/sec) (average 2945095.8 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (2745380.8 KiBytes/sec) (average 2798462.2 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (3117180.9 KiBytes/sec) (average 2746070.8 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (3117180.9 KiBytes/sec) (average 2844253.7 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (3007341.7 KiBytes/sec) (average 2878659.9 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (2519064.0 KiBytes/sec) (average 2956651.4 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (3093655.5 KiBytes/sec) (average 2894340.3 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (2820772.0 KiBytes/sec) (average 2732566.5 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (2773312.0 KiBytes/sec) (average 2790907.3 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (3131490.8 KiBytes/sec) (average 2864041.8 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (3131490.8 KiBytes/sec) (average 2814900.0 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (2959060.9 KiBytes/sec) (average 2942927.2 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (3083655.5 KiBytes/sec) (average 2937393.2 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (2970743.3 KiBytes/sec) (average 2879300.8 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (3083655.5 KiBytes/sec) (average 2895262.7 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (2824827.2 KiBytes/sec) (average 2873199.6 KiBytes/sec)
getting file %S0.dat of size 2097152000 as /dev/null (3228580.0 KiBytes/sec) (average 2873199.6 KiBytes/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, 0.0 id, 0.0 wa, 0.2 hi, 2.1 si, 0.0 st
MiB Mem: 191824.1 total, 170025.4 free, 3318.7 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 O CPU MEMER TIME+ COMMAND
322763 root 20 0 376228 36620 17364 R 0.0 2.0 1:26.20 smbclient
322764 root 20 0 360836 28192 17120 R 0.15 0.0 1:26.18 smbclient
322765 root 20 0 360040 28536 17164 R 0.1 0.0 1:25.16 smbclient
322760 root 20 0 376244 36740 17468 R 79.0 0.0 1:23.73 smbclient
322762 root 20 0 376236 36700 17228 R 79.0 0.0 1:24.42 smbclient
322761 root 20 0 376248 36908 17292 R 79.5 0.0 1:24.74 smbclient
322766 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 0 0 0 0 23.0 0.0 0:14.04 io_wqe_worker-0
322827 root 20 0 0 0 0 0 23.5 0.0 0:12.77 io_wqe_worker-0
322882 root 20 0 0 0 0 0 22.8 0.0 0:12.36 io_wqe_worker-0
322838 root 20 0 0 0 0 0 22.8 0.0 0:12.96 io_wqe_worker-0
322772 root 20 0 458260 21600 17596 R 22.5 0.0 0:22.49 smbdc
322796 root 20 0 376248 36908 17292 R 79.5 0.0 0:14.00 io_wqe_worker-0
322880 root 20 0 0 0 0 0 22.5 0.0 0:14.13 io_wqe_worker-0
322822 root 20 0 0 0 0 0 21.25 0.0 0:12.86 io_wqe_worker-0
322818 root 20 0 0 0 0 0 19.2 0.0 0:12.71 io_wqe_worker-0
318818 root 20 0 246476 6976 4980 S 9.3 0.0 1:31.29 iftop
322833 root 20 0 0 0 0 0 8.5 0.0 0:02.78 io_wqe_worker-0
322854 root 20 0 0 0 0 0 5.0 0.0 0:02.50 io_wqe_worker-0
322842 root 20 0 0 0 0 0 5.4 0.0 0:02.70 io_wqe_worker-0
322861 root 20 0 0 0 0 0 4.6 0.0 0:02.46 io_wqe_worker-0
322860 root 20 0 0 0 0 0 4.6 0.0 0:02.54 io_wqe_worker-0
322862 root 20 0 0 0 0 0 4.6 0.0 0:02.78 io_wqe_worker-0
310730 root 20 0 3837104 172756 54944 S 4.3 0.1 1:49.89 perf
322836 root 20 0 0 0 0 0 4.3 0.0 0:02.61 io_wqe_worker-0
322839 root 20 0 0 0 0 0 4.3 0.0 0:02.77 io_wqe_worker-0
322846 root 20 0 0 0 0 0 4.0 0.0 0:02.52 io_wqe_worker-0
322865 root 20 0 0 0 0 0 4.0 0.0 0:02.60 io_wqe_worker-0
322868 root 20 0 0 0 0 0 4.0 0.0 0:02.46 io_wqe_worker-0
322887 root 20 0 0 0 0 0 4.0 0.0 0:02.57 io_wqe_worker-0
322845 root 20 0 0 0 0 0 3.6 0.0 0:02.58 io_wqe_worker-0
322856 root 20 0 0 0 0 0 3.6 0.0 0:02.33 io_wqe_worker-0
322858 root 20 0 0 0 0 0 3.6 0.0 0:02.52 io_wqe_worker-0

Samples: 30M of event 'cycles', 1000 Hz, Event count (approx.): 52678550929 Lost: 0/0 drop: 0/0
Overhead Shared object Symbol
0.40% [kernel] [k] copy_user_enhanced_fast_string
0.40% [kernel] [k] native_queued_spin_lock_slowpath
7.30% [kernel] [k] tcpackot_rcv
1.79% [kernel] [k] do_tcp_sendpages
1.20% [kernel] [k] raw_spin_lock_bh
1.23% [kernel] [k] prb_fill_curr_block.isra.0
1.03% [kernel] [k] raw_spin_lock
0.92% [kernel] [k] copy_page_to_user
0.89% [kernel] [k] skb_release_data
0.89% [kernel] [k] check_object_size

Tx: cum: 226476B peak: 6.596B rates: 1816b 1816b 1866b
Rx: 4B 0B 0B 0B 0B 0B
TOTAL: 226476B 6.596B 1816b 1816b 1866b
```



Stefan Metzmacher

io_uring (14/21)



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

- ▶ We need support for TEVENT_FD_ERROR in order to monitor errors
 - ▶ When using IORING_OP_SEND,RECVMSG we still want to notice errors
 - ▶ This is the main merge request:
 - ▶ https://gitlab.com/samba-team/samba/-/merge_requests/2793
 - ▶ This merge request converts Samba to use TEVENT_FD_ERROR:
 - ▶ https://gitlab.com/samba-team/samba/-/merge_requests/2885
 - ▶ (It also simplifies other places in the code without io_uring)

The road to upstream (samba_io_uring abstraction 1)

API glue to tevent:

```
void samba_io_uring_ev_register(void);

const struct samba_io_uring_features *samba_io_uring_system_features(void);

struct samba_io_uring *samba_io_uring_ev_context_get_ring(struct tevent_context *ev);

const struct samba_io_uring_features *samba_io_uring_get_features(
    const struct samba_io_uring *ring);

ev = tevent_context_init_byname(mem_ctx, "samba_io_uring_ev");
```

- ▶ samba_io_uring abstraction factored out of vfs_io_uring:
 - ▶ samba_io_uring_ev_hybrid tevent backend (glued on epoll backend)
 - ▶ It means every layer getting the tevent_context can use io_uring
 - ▶ No #ifdef's just checking if the required features are available

The road to upstream (samba_io_uring abstraction 1)

API glue to tevent:

```
void samba_io_uring_ev_register(void);

const struct samba_io_uring_features *samba_io_uring_system_features(void);

struct samba_io_uring *samba_io_uring_ev_context_get_ring(struct tevent_context *ev);

const struct samba_io_uring_features *samba_io_uring_get_features(
    const struct samba_io_uring *ring);

ev = tevent_context_init_byname(mem_ctx, "samba_io_uring_ev");
```

- ▶ samba_io_uring abstraction factored out of vfs_io_uring:
 - ▶ samba_io_uring_ev_hybrid tevent backend (glued on epoll backend)
 - ▶ It means every layer getting the tevent_context can use io_uring
 - ▶ No #ifdef's just checking if the required features are available

The road to upstream (samba_io_uring abstraction 2)

generic submission/completion api:

```
void samba_io_uring_completion_prepare(struct samba_io_uring_completion *completion,
                                       void (*completion_fn)(struct samba_io_uring_completion *completion,
                                                             void *completion_private,
                                                             const struct io_uring_cqe *cqe),
                                       void *completion_private);

void samba_io_uring_submission_prepare(struct samba_io_uring_submission *submission,
                                       void (*submission_fn)(struct samba_io_uring *ring,
                                                             struct samba_io_uring_submission *submission,
                                                             void *submission_private),
                                       void *submission_private,
                                       struct samba_io_uring_completion *completion);

struct io_uring_sqe *samba_io_uring_submission_sqe(struct samba_io_uring_submission *
                                                  submission);

size_t samba_io_uring_queue_submissions(struct samba_io_uring *ring,
                                         struct samba_io_uring_submission *submission);
```

▶ Using it ...

- ▶ convert `vfs.io_uring`
- ▶ use it in `smb2_server.c`
- ▶ In future use it in other performance critical places too.

The road to upstream (samba_io_uring abstraction 2)

generic submission/completion api:

```
void samba_io_uring_completion_prepare(struct samba_io_uring_completion *completion,
    void (*completion_fn)(struct samba_io_uring_completion *completion,
        void *completion_private,
        const struct io_uring_cqe *cqe),
    void *completion_private);

void samba_io_uring_submission_prepare(struct samba_io_uring_submission *submission,
    void (*submission_fn)(struct samba_io_uring *ring,
        struct samba_io_uring_submission *submission,
        void *submission_private),
    void *submission_private,
    struct samba_io_uring_completion *completion);

struct io_uring_sqe *samba_io_uring_submission_sqe(struct samba_io_uring_submission *
    submission);

size_t samba_io_uring_queue_submissions(struct samba_io_uring *ring,
    struct samba_io_uring_submission *submission);
```

▶ Using it ...

- ▶ convert vfs_io_uring
- ▶ use it in smb2_server.c
- ▶ In future use it in other performance critical places too.

The road to upstream (smb2_server.c)

- ▶ Refactoring of smb2_server.c
 - ▶ add optional IORING_OP_SENDMSG, IORING_OP_RECVMSG support
- ▶ There are structural problems with splice from a file
 - ▶ I had a discussion with the Linux developers about it:
 - ▶ The page content from the page cache may change unexpectedly
 - ▶ <https://lists.samba.org/archive/samba-technical/2023-February/thread.html#137945>
 - ▶ We may not be able to use IORING_OP_SENDMSG/SPLICE by default
 - ▶ Maybe IORING_OP_RECVMSG/SPLICE is possible
- ▶ With IORING_OP_SENDMSG_ZC only 1 one copy is used:
 - ▶ It is able to avoid copying to the socket
 - ▶ We get an extra completion once the buffers are not needed anymore
 - ▶ Only with real hardware, not on loopback in an upstream kernel
 - ▶ A custom kernel loopback gives ~7.5 GBytes/s instead of ~3.5 GBytes/s
 - ▶ With a noop vfs module, we get ~18 GBytes/s instead of ~6 GBytes/s

The road to upstream (smb2_server.c)

- ▶ Refactoring of smb2_server.c
 - ▶ add optional IORING_OP_SENDMSG, IORING_OP_RECVMSG support
- ▶ There are structural problems with splice from a file
 - ▶ I had a discussion with the Linux developers about it:
 - ▶ The page content from the page cache may change unexpectedly
 - ▶ <https://lists.samba.org/archive/samba-technical/2023-February/thread.html#137945>
 - ▶ We may not be able to use IORING_OP_SENDMSG/SPLICE by default
 - ▶ Maybe IORING_OP_RECVMSG/SPLICE is possible
- ▶ With IORING_OP_SENDMSG_ZC only 1 one copy is used:
 - ▶ It is able to avoid copying to the socket
 - ▶ We get an extra completion once the buffers are not needed anymore
 - ▶ Only with real hardware, not on loopback in an upstream kernel
 - ▶ A custom kernel loopback gives ~7.5 GBytes/s instead of ~3.5 GBytes/s
 - ▶ With a noop vfs module, we get ~18 GBytes/s instead of ~6 GBytes/s

The road to upstream (smb2_server.c)

- ▶ Refactoring of smb2_server.c
 - ▶ add optional IORING_OP_SENDMSG, IORING_OP_RECVMSG support
- ▶ There are structural problems with splice from a file
 - ▶ I had a discussion with the Linux developers about it:
 - ▶ The page content from the page cache may change unexpectedly
 - ▶ <https://lists.samba.org/archive/samba-technical/2023-February/thread.html#137945>
 - ▶ We may not be able to use IORING_OP_SENDMSG/SPLICE by default
 - ▶ Maybe IORING_OP_RECVMSG/SPLICE is possible
- ▶ With IORING_OP_SENDMSG_ZC only 1 copy is used:
 - ▶ It is able to avoid copying to the socket
 - ▶ We get an extra completion once the buffers are not needed anymore
 - ▶ Only with real hardware, not on loopback in an upstream kernel
 - ▶ A custom kernel loopback gives ~7.5 GBytes/s instead of ~3.5 GBytes/s
 - ▶ With a noop vfs module, we get ~18 GBytes/s instead of ~6 GBytes/s

- ▶ Patches are slowly getting prepared for master
 - ▶ Some preparations are already in or pending merge requests
 - ▶ We even have basic automated ci testing in place now
 - ▶ But changes need to be checked for performance regressions
- ▶ We can use `io_uring` deep inside of the `smbclient` code
 - ▶ The low layers can just use `samba_io_uring_ev_context_get_ring()`
 - ▶ And use it available without changing the whole stack

- ▶ Patches are slowly getting prepared for master
 - ▶ Some preparations are already in or pending merge requests
 - ▶ We even have basic automated ci testing in place now
 - ▶ But changes need to be checked for performance regressions
- ▶ We can use `io_uring` deep inside of the `smbclient` code
 - ▶ The low layers can just use `samba_io_uring_ev_context_get_ring()`
 - ▶ And use it available without changing the whole stack

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

→ SerNet/SAMBA+ sponsor booth

Slides: <https://samba.org/~metze/presentations/2023/SDC/>