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 an updated version of this presentation here:

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

- What is SMB3 Multichannel?
- Updates in Samba 4.15
- What is io-uring?
- io-uring for Samba
- Performance research, prototyping and ideas
- Questions? Feedback!
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 ”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)
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
In October 2020 I was able to do some performance research
  With 100GBit/s interfaces and two NUMA nodes per server.

At that time I focussed on the SMB2 Read performance only
  We had limited time on the given hardware
  We mainly tested with fio.exe on a Windows client
  Linux kernel 5.8.12 on the server

More verbose details can be found here:
  https://lists.samba.org/archive/samba-technical/2020-October/135856.html
Performance with MultiChannel, sendmsg()

4 connections, ~3.8 GBytes/s, bound by >500% cpu in total, sendmsg() takes up to 0.5 msecs
IORING_OP_SENDMSG (Part1)

4 connections, ~6.8 GBytes/s, smbd only uses ~11% cpu, (io_wqe_work ~50% cpu) per connection, we still use >300% cpu in total

top - 05:45:38 up 2 days, 46 min, 2 users, load average: 3.03, 2.04, 1.01
threads: 823 total, 3 running, 820 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.1 us, 4.7 sy, 0.0 ni, 96.1 id, 0.0 wa, 0.1 hi, 0.5 sl, 0.0 st
Mem: 191924.1 total, 102194.6 free, 2702.6 used, 6726.9 buff/cache
Swap: 1024.0 total, 1024.0 free, 0.0 used, 105594.7 avail Mem

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PRI</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR</th>
<th>S</th>
<th>%CPU</th>
<th>%MEM</th>
<th>TIME+</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>307577</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>R</td>
<td>49.0</td>
<td>0.0</td>
<td>0:05:00</td>
<td>io_wqe_worker-0</td>
</tr>
<tr>
<td>307549</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>46.0</td>
<td>0.0</td>
<td>0:21:39</td>
<td>io_wqe_worker-0</td>
</tr>
<tr>
<td>307555</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>44.0</td>
<td>0.0</td>
<td>0:21:45</td>
<td>io_wqe_worker-0</td>
</tr>
<tr>
<td>307567</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>29.8</td>
<td>0.0</td>
<td>0:09:92</td>
<td>io_wqe_worker-1</td>
</tr>
<tr>
<td>307558</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>663100</td>
<td>144024</td>
<td>18084</td>
<td>S</td>
<td>23.2</td>
<td>0.1</td>
<td>0:09:10</td>
<td>smbd</td>
</tr>
<tr>
<td>307556</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>663100</td>
<td>144024</td>
<td>18084</td>
<td>S</td>
<td>19.9</td>
<td>0.1</td>
<td>0:09:95</td>
<td>smbd</td>
</tr>
<tr>
<td>307559</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>663100</td>
<td>144024</td>
<td>18084</td>
<td>S</td>
<td>19.5</td>
<td>0.1</td>
<td>0:09:92</td>
<td>smbd</td>
</tr>
<tr>
<td>307563</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>663100</td>
<td>144024</td>
<td>18084</td>
<td>S</td>
<td>19.6</td>
<td>0.1</td>
<td>0:09:86</td>
<td>smbd</td>
</tr>
<tr>
<td>307557</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>663100</td>
<td>144024</td>
<td>18084</td>
<td>S</td>
<td>19.2</td>
<td>0.1</td>
<td>0:09:31</td>
<td>smbd</td>
</tr>
<tr>
<td>307560</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>663100</td>
<td>144024</td>
<td>18084</td>
<td>S</td>
<td>19.2</td>
<td>0.1</td>
<td>0:09:38</td>
<td>smbd</td>
</tr>
<tr>
<td>307561</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>663100</td>
<td>144024</td>
<td>18084</td>
<td>S</td>
<td>19.2</td>
<td>0.1</td>
<td>0:09:31</td>
<td>smbd</td>
</tr>
<tr>
<td>307534</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>663100</td>
<td>144024</td>
<td>18084</td>
<td>S</td>
<td>18.9</td>
<td>0.1</td>
<td>0:09:06</td>
<td>smbd</td>
</tr>
<tr>
<td>307576</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>663100</td>
<td>144024</td>
<td>18084</td>
<td>S</td>
<td>18.9</td>
<td>0.1</td>
<td>0:05:61</td>
<td>smbd</td>
</tr>
<tr>
<td>307562</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>663100</td>
<td>144024</td>
<td>18084</td>
<td>S</td>
<td>18.9</td>
<td>0.1</td>
<td>0:09:33</td>
<td>smbd</td>
</tr>
<tr>
<td>307530</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>663100</td>
<td>144024</td>
<td>18084</td>
<td>D</td>
<td>11.3</td>
<td>0.1</td>
<td>0:05:16</td>
<td>smbd</td>
</tr>
<tr>
<td>307552</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>9.3</td>
<td>0.0</td>
<td>0:12:25</td>
<td>io_wqe_worker-0</td>
</tr>
<tr>
<td>417</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.3</td>
<td>0.0</td>
<td>0:05:50</td>
<td>kworker/0:2:events</td>
</tr>
<tr>
<td>307583</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.3</td>
<td>0.0</td>
<td>0:09:61</td>
<td>kworker/1:0:2:all</td>
</tr>
<tr>
<td>307560</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.3</td>
<td>0.0</td>
<td>0:09:02</td>
<td>kworker/29:0:events</td>
</tr>
<tr>
<td>307580</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>62964</td>
<td>5532</td>
<td>3044</td>
<td>R</td>
<td>0.3</td>
<td>0.0</td>
<td>0:00:12</td>
<td>top</td>
</tr>
<tr>
<td>1</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>242512</td>
<td>10952</td>
<td>8176</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:02:04</td>
<td>systemd</td>
</tr>
<tr>
<td>2</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00:13</td>
<td>kthreadd</td>
</tr>
<tr>
<td>3</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>-20</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00:00</td>
<td>rcu_gp</td>
</tr>
<tr>
<td>4</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>-20</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00:00</td>
<td>rcu_par_gp</td>
</tr>
<tr>
<td>6</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>-20</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00:00</td>
<td>kworker/0:0:h-kblob</td>
</tr>
<tr>
<td>10</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>-20</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00:00</td>
<td>mm_percpu.wq</td>
</tr>
<tr>
<td>11</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>-20</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00:00</td>
<td>kworker/0:0:h-kblob</td>
</tr>
<tr>
<td>12</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>-20</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00:00</td>
<td>rcu_sched</td>
</tr>
<tr>
<td>13</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00:00</td>
<td>migration/0</td>
</tr>
<tr>
<td>14</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00:00</td>
<td>cpuprev/0</td>
</tr>
<tr>
<td>15</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00:00</td>
<td>cpuprev/0</td>
</tr>
<tr>
<td>16</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00:00</td>
<td>migthread/0</td>
</tr>
<tr>
<td>17</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00:00</td>
<td>migthread/0</td>
</tr>
<tr>
<td>19</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>-20</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00:00</td>
<td>kworker/1:0:h-kblob</td>
</tr>
<tr>
<td>21</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>-20</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00:00</td>
<td>cpuprev/0</td>
</tr>
<tr>
<td>22</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>-20</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00:00</td>
<td>migthread/0</td>
</tr>
<tr>
<td>23</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>-20</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00:00</td>
<td>migthread/0</td>
</tr>
<tr>
<td>24</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>-20</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00:00</td>
<td>kworker/1:0:h-kblob</td>
</tr>
<tr>
<td>26</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00:00</td>
<td>cpuprev/0</td>
</tr>
<tr>
<td>27</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00:00</td>
<td>migthread/0</td>
</tr>
</tbody>
</table>
IORING_OP_SENDMSG (Part2)

The results vary heavily depending on the NUMA bouncing, between 5.0 GBytes/s and 7.6 GBytes/s
IORING_OP_SENDMSG (Part3)

The major problem still exists, memory copy done by copy_user_enhanced_fast_string()
IORING_OP_SENDMSG + IORING_OP_SPLICE (Part1)

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

The Windows client was still the bottleneck with "Set-SmbClientConfiguration -ConnectionCountPerRssNetworkInterface 16"
smbclient is the bottleneck here too
smbclient IORING_OP SENDMSG/SPLICE (loopback)

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

smbclient is the bottleneck here too, it triggers the memory copy done by copy_user_enhanced_fast_string()
More loopback testing on brand new hardware

- Recently I re-did the loopback read tests with 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

Stefan Metzmacher
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
    - (vmssplice 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
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/