io_uring
Status Update within Samba

Stefan Metzmacher <metze@samba.org>
Samba Team / SerNet

2023-09-20

https://samba.org/~metze/presentations/2023/SDC/
Topics

▶ 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 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
  - 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)
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
  - 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

Stefan Metzmacher
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](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:
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

Stefan Metzmacher io_uring (10/21)
IORING_OP_SENDMSG (Part2)

The major problem still exists, memory copy done by copy_user_enhanced_fast_string()
IORING_OP_SENDMSG + IORING_OP_SPLIT (Part 1)

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

The Windows client was still the bottleneck with "Set-SmbClientConfiguration -ConnectionCountPerRssNetworkInterface 16"
smbclient IORING_OP_SENDMSG/SPLICE (network)

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

smbclient is the bottleneck here too
smbclient IORING_OP SENDMSG/SPLICE (loopback)

8 connections, ~22 GBytes/s, smbd 22% cpu, with 4 io_wq_worker_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
  - 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
io_uring (15/21)
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
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:

```c
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
API glue to tevent:

```c
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:

```c
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.

Stefan Metzmacher  io_uring (18/21)
The road to upstream (samba io_uring abstraction 2)

generic submission/completion api:

```c
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](https://lists.samba.org/archive/samba-technical/2023-February/thread.html#137945)
  - We may not 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

Stefan Metzmacher io_uring (19/21)
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](https://lists.samba.org/archive/samba-technical/2023-February/thread.html#137945)
  - We may not 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 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
Future Improvements

- 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 if available without changing the whole stack
Future Improvements

▶ 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 if available without changing the whole stack
Questions? Feedback!

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

→ SerNet/SAMBA+ sponsor booth

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