Tuning Samba for SMB3 load

Volker Lendecke

May 19, 2014

SMB3 has been designed by Microsoft for Datacenter use. The main focus of SMB3 is to provide performance and reliability equal to traditional SAN architectures. Two workloads have been pointed out in several Microsoft presentations as very important: HyperV images and SQL database files. Both workloads put full focus on pure read and write requests. These files are created and opened once in a session and from there on used with random I/O requests.

We assert that smbd is very good at handling this load today and is flexible enough to be adapted to specialized kernel APIs that support high-speed low-latency APIs for network servers.

1 smbd architecture

Samba is a Unix daemon with a typical architecture for Unix daemons: One process listens on a network socket and every client connection is handled by a separate process.

This architecture has helped Samba in many aspects:

Fault tolerance: When an smbd dies, only the one client that this smbd was serving is affected. Samba has mechanisms to deal with stale data in shared databases.

Scalability: Due to the shared-nothing approach of multiple processes, all interfaces between smbds that are required to implement SMB performance are explicit. Contention points are more easily isolated and fixed than with a shared-memory architecture.

Cluster awareness: Because of the explicit interfaces mentioned, it was relatively easy to identify all spots in the code that need to be replicated in a cluster environment.

Historically, SMB clients were single threaded. Samba matches this single threaded client behaviour: Every smbd serving a client is a single threaded process. With recent SMB versions, clients have changed: More and more applications issue multiple requests simultaneously and expect the server to handle them in parallel. Samba has already good support for fully supporting this parallel client behaviour for the important Read/Write

path: It has been demonstrated many years ago in a customer environment that Samba can easily saturate a 10GigE link on top of a clustered file system accessing just a single file over one single SMB1 TCP connection given proper parallel client behaviour.

With SMB3 clients, this proper parallel client behaviour becomes the default behaviour and is not exceptional anymore.

2 Optimizing READ/WRITE

Samba has implemented asynchronous I/O for the Read/Write path with Samba 3.2 in 2008. Since then, Samba is prepared for handling multiple simultaneous client requests and passes them on to kernel pread/pwrite calls in parallel. Samba has done three implementations of this, based on practical experiences with the different approaches.

Posix AIO: The System V Unix Specification contains an API for asynchronous I/O based on calls like aio_read, aio_write, aio_return and others. This is the basis for the initial implementation Samba has taken and is still around for Unix systems that have proper kernel-level support for this API family.

Linux does not have kernel level support for Posix AIO, the glibc implements this in user space based on top of pthreads: Every I/O request is put into a queue and a pool of helper threads do the actual pread/pwrite calls. The glibc implementation however has two severe deficiencies:

- It implements the Posix API, which is based on signals. For Samba, this has caused quite some trouble.
- Glibc serializes multiple I/O requests for a single file descriptor.

In particular the latter deficiency is a major hurdle for Samba to properly implement a high-performance server serving HyperV images.

aio_fork The Samba Team, in particular Andrew Tridgell, had to discover that the Linux support for Posix AIO is not only a bad workaround for the lack of a kernel API, it is outright broken. Posix AIO completion notification uses real time signals. Those real time signals together with smbd which changes user credentials is flaky at best, and the RHEL kernel used at the time when this was heavily been tested had clear bugs by completely losing those signals.

The Samba Team, in particular Volker Lendecke from SerNet, responded to these problems by implementing the AIO functionality based on processes communicating via sockets and shared memory. This is more heavy-weight than a proper kernel interface, but it proved to be very solid and good enough for the job at the time.

aio_pthread The aio_fork approach, while being very robust, is much too heavy-weight for general use. With Samba 4.0 in 2012 Volker Lendecke from SerNet has implemented Async I/O based on pthreads with an API that matches Samba's internal concurrent architecture better.

Since the implementation of the aio_pthread approach a lot of tuning work has been put into speeding up read/write. The main focus is to spend as little time as possible between reading a client request from the network socket and passing it on to the kernel pread/pwrite system call. In the recent past, a vendor of a small embedded SoC has sponsored significant work in this area: Embedded CPUs usually are very slow compared to the network adapters, and Samba can now saturate a Gigabit link with a small embedded CPU. Most of this work to speed up the SMB2 read/write path was done by Stefan Metzmacher as part of is work for SerNet.

High-performance I/O directly benefits from this work, because it significantly reduced the time spent in user space between the network socket and the pread/pwrite calls and thus made it possible to more easily serve lots of parallel requests.

3 Multichannel

SMB3 brings a feature called Multichannel. With Multichannel, the client will open multiple TCP connections for the same logical SMB session. This enables load-balancing across multiple network adapters without the hassle of network bonding. Also, it helps to more swiftly survive short glitches on individual physical network links.

Samba will implement multichannel by making one smbd serve all TCP connections belonging to one logical SMB session. The initial implementation will follow the standard smbd architecture: One thread polls for all network connections, reads the data from the network and passes the R/W requests to the async pread/pwrite helper threads with minimal overhead.

We are fully aware that architecture has its potential limitation: One single dispatcher process might turn out to be a bottleneck if it has to serve many TCP connections. There have been discussions and initial design work to completely overcome this limitation: A Samba vendor wants to instrument the kernel to handle SMB2 read and write requests completely in the kernel space. The network driver or some filter module in the kernel will inspect the incoming TCP data and pick out the requests it can handle and not pass these requests on to the smbd in user space at all, avoiding the syscall overhead completely for these requests. To implement this properly and to develop the necessary interfaces, it is necessary to have a multi-process user space implementation of this concept: One process polls for the network sockets and picks the SMB2 read and write requests it can handle. It passes on everything else to the traditional smbd. If this architecture is implemented using multiple processes, it will be easily possible to put that task into a thread, much like it has happened with the transition of aio_fork to aio_pthread.

There is precedence in Samba already for this style of multiple process handling a single SMB session: To overcome long blocking file system calls like unlink, Samba implements the so-called "async smb echo responder", a helper process which independently replies to SMB-level echo requests. Clients start sending those echo requests when individual SMB requests like SMBunlink take longer than a few seconds and use these echo requests as a liveness test of the SMB connection. This concept can be extended to handle SMB2

read/write requests in independent processes. The async smb echo responder, which could be used as a blueprint for the SMB2 read/write threads was done together by Stefan Metzmacher and Volker Lendecke.

4 Zero-Copy

Samba does implement zero-copy for the read path using the sendfile API. However, the sendfile API monopolizes the network socket while the file system is busy reading blocks from the disk.

Samba has implemented prototype code for async zero-copy I/O using the Linux splice API. However, it turned out that the splice API does not improve performance when compared to network read to user space and threaded pwrite from user space to disk. So Samba has no code in production for async zero-copy I/O, but mainly due to the lack of proper Linux API support for this. If someone gave Samba a good API to support async zero-copy I/O, we would be more than happy to implement this. The smbd read/write code path is flexible enough to be easily accommodated to such an API.

A few Samba OEMs have a custom Linux kernel patch in place to improve recvfile. Samba now has patches to properly work with these modified kernels.

5 RDMA

Microsoft Windows 2012 implements SMB over RDMA. RDMA can serve very high data throughput, very low latency with very few CPU cycles. Stefan Metzmacher from SerNet has done prototype code to do SMB over RDMA in Samba. When he did it, he found several deficiencies in both the RDMA kernel drivers as well as the supporting user-space libraries that at the time made it difficult to properly implement RDMA in Samba. With cooperation from RDMA developers it should however be no major problem to adapt Samba to offer SMB services over RDMA.

A first implementation of RDMA in Samba might not do the full zero-copy path: The SMB requests are RDMA-transferred into smbd user space and then with pread/pwrite transferred to disk. An alternative implementation might mmap the file in question and directly RDMA into the mmap buffer. It depends on the file system APIs how many or rather how few memory copies have to happen before client data actually ends up on rotating rust or in flash pages.