HOW I LEARNED TO LOVE PERF

...AND SYSTEMTAP
1. Introduction: understanding Samba fileserver performance
   1.1. Case study: cp 10k 10 KB files
2. Performance metrics: tools / tracing frameworks
   2.1. /proc based tools (ps, pidstat, top, vmstat, iostat, …)
   2.2. Systemtap
   2.3. perf
3. Demo: perf
4. Recent performance improvements in smbd
1.1. CASE STUDY

- Small files copy: 10,000 files, each 10 KB
- Time for local copy is under one second at nearly 100% CPU:

```
$ find testfiles/ -type d | wc -l
101

$ find testfiles/ -type f | wc -l
10000

$ du -sh testfiles/
118M    testfiles/

$ time cp -r testfiles/ copy
real    0m0.294s
user    0m0.016s
sys     0m0.269s
```
1.1. CASE STUDY

- Copy to Linux kernel client mount (SMB2)
- Start copy, wait…. and wait…
- `pidstat` shows `smbd` using more than 50% CPU:

```bash
$ pidstat -p 20182 1
Linux 4.10.12-200.fc25.x86_64+debug (kazak) 04/30/2017 _x86_64_

<table>
<thead>
<tr>
<th>Time</th>
<th>UID</th>
<th>PID</th>
<th>%usr</th>
<th>%system</th>
<th>%guest</th>
<th>%CPU</th>
<th>CPU</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>03:46:13 PM</td>
<td>1000</td>
<td>20182</td>
<td>22.00</td>
<td>38.00</td>
<td>0.00</td>
<td>60.00</td>
<td>2</td>
<td>smbd</td>
</tr>
<tr>
<td>03:46:14 PM</td>
<td>1000</td>
<td>20182</td>
<td>23.00</td>
<td>37.00</td>
<td>0.00</td>
<td>60.00</td>
<td>2</td>
<td>smbd</td>
</tr>
<tr>
<td>03:46:15 PM</td>
<td>1000</td>
<td>20182</td>
<td>25.00</td>
<td>36.00</td>
<td>0.00</td>
<td>61.00</td>
<td>2</td>
<td>smbd</td>
</tr>
<tr>
<td>03:46:16 PM</td>
<td>1000</td>
<td>20182</td>
<td>22.00</td>
<td>39.00</td>
<td>0.00</td>
<td>61.00</td>
<td>2</td>
<td>smbd</td>
</tr>
<tr>
<td>03:46:17 PM</td>
<td>1000</td>
<td>20182</td>
<td>26.00</td>
<td>32.00</td>
<td>0.00</td>
<td>58.00</td>
<td>2</td>
<td>smbd</td>
</tr>
<tr>
<td>03:46:18 PM</td>
<td>1000</td>
<td>20182</td>
<td>28.00</td>
<td>30.00</td>
<td>0.00</td>
<td>58.00</td>
<td>3</td>
<td>smbd</td>
</tr>
<tr>
<td>03:46:19 PM</td>
<td>1000</td>
<td>20182</td>
<td>23.00</td>
<td>33.00</td>
<td>0.00</td>
<td>56.00</td>
<td>3</td>
<td>smbd</td>
</tr>
<tr>
<td>03:46:20 PM</td>
<td>1000</td>
<td>20182</td>
<td>21.00</td>
<td>29.00</td>
<td>0.00</td>
<td>50.00</td>
<td>3</td>
<td>smbd</td>
</tr>
<tr>
<td>03:46:21 PM</td>
<td>1000</td>
<td>20182</td>
<td>18.00</td>
<td>32.00</td>
<td>0.00</td>
<td>50.00</td>
<td>1</td>
<td>smbd</td>
</tr>
<tr>
<td>^C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average:</td>
<td>1000</td>
<td>20182</td>
<td>23.11</td>
<td>34.00</td>
<td>0.00</td>
<td>57.11</td>
<td></td>
<td>smbd</td>
</tr>
</tbody>
</table>
```
1.1. CASE STUDY

- takes 13 seconds, so roughly 50x slower:

```
$ mount | grep cifs
//localhost/share on /mnt type cifs
(rw,nosuid,nodev,relatime,vers=2.1, ...)

$ time cp -r testfiles/ /mnt/copy_testfiles/
real       0m13.399s
user       0m0.028s
sys        0m1.474s
```
Some background info on the system under test:

- Idle system, no other noticeable CPU consumers
- Linux system in Virtualbox, 4 cores
- SSD disk
- mount over loopback network device
1.1. CASE STUDY

What the hell is smbd is doing:

1. Processing data (and so using the CPU)

2. Waiting for data (disk or network)

And it turns out in this case we’re actually 50% waiting

50% of 13s means 6.5 s. Good heavens!

SMB(2): request / response protocol and for many operations the client needs the result before proceeding (eg opening a file handle) and can’t progress ⇒ latency
1.1. CASE STUDY

SMB op latency

- Client sends request
- Server received request
- Server sends response
- Client receives response
- Client sends next request

Time

- Server receives request
- Processing
- Server sends response
- Waiting
- Server received request
1.1. CASE STUDY

- The client *could* send other requests, but it doesn’t
- So this explains half of the desaster
- Oh, and remember: I’m interested in per client throughput, not system throughput – for system throughput just add more clients (and possibly more CPUs)
- Is there another way of inferring wait time without interpolating from top/pidstat?
- Let’s look at the performance metrics tools available on Linux (this is 2017, not 2007)
It’s Tool Time!

LINUX PERFORMANCE OBSERVABILITY TOOLS
2. TOOLS & KERNEL FRAMEWORKS

Linux Performance Observability Tools

HTTP://WWW.BRENDA Gregg.COM/Linuxperf.html 2017
2. KERNEL FRAMEWORKS: OLD

- ftrace
- perf_events
- eBPF
- SystemTap
- LTTng
- ktap
- dtrace4linux
- OEL DTrace
- sysdig

/proc
2. KERNEL FRAMEWORKS: ALL

- ftrace
- perf_events
- eBPF
- SystemTap
- LTTng
- ktap
- dtrace4linux
- OEL DTrace
- sysdig

/proc
2. KERNEL FRAMEWORKS: NEW

- ftrace
- perf_events
- eBPF
- SystemTap
- LTTng
- ktap
- dtrace4linux
- OEL DTrace
- sysdig

/proc
2.1. /PROC BASED TOOLS

- /proc based tools, let's try one

- Waiting for network or disk can be traced with `strace`, example:
2.1. /PROC BASED TOOLS

- Quite nice, but copy time goes up the roof: 150 s
- Even though tracing is limited: pread/pwrite, readv/writev and epoll_wait
- iow: strace sucks (hey Jeremy!), but at least it’s documented in man strace(1): 
  BUGS: A traced process runs slowly.
Anything else?

Requirement: near zero overhead tracer, at least for syscalls, but more (user-space functions) is better

The answer (at least on Linux):

systemtap
- "SystemTap provides a simple command line interface and scripting language for writing instrumentation for a live running kernel plus user-space applications."

- "The essential idea behind … systemtap … is to name events, and to give them handlers. Whenever a specified event occurs, the Linux kernel runs the handler."

- *You* write the handlers in the Systemtap script language (C like, but safe)

- Can trace everything and profile data can be aggregated any way you like

- Requires kernel-debuginfo for diving into the kernel, but (hopefully) not needed for instrumenting user-space with static-probes
So two things:

1. Trace points / events / probes:
   - Trace points are no-ops when not enabled, low overhead when enabled
   - Many existing trace points in the kernel (man stapprobes)
   - For user-space applications add „static probes“ in the source code
   - With DWARF debug symbols it’s even possible to use source lines as trace points

2. Scripts in the Systemtap script language
   - Scripts provide handlers for events
   - Scripts are translated to C and then compiled to create a kernel module
   - When the module is loaded, it activates all the requested events
So let's apply this to smbd and add static probes.
commit dc82855eedf84fb963cf71ddf2e2f5e9343d6426
Author:     Ralph Boehme <slow@samba.org>
AuthorDate: Sun Apr 23 11:53:47 2017 +0200
Commit:     Ralph Boehme <slow@samba.org>
CommitDate: Fri Apr 28 11:34:30 2017 +0200

s3/smbd: add SMB2 request probe points

---

source3/smbd/smb2_server.c | 9 ++++++++++
1 file changed, 9 insertions(+)

diff --git a/source3/smbd/smb2_server.c b/source3/smbd/smb2_server.c
index cf5be6a..e00d1a4 100644
--- a/source3/smbd/smb2_server.c
+++ b/source3/smbd/smb2_server.c
@@ -32,6 +32,8 @@
 #include "auth.h"
 #include "lib/crypto/sha512.h"

+#include <sys/sdt.h>
+
 static void smbd_smb2_connection_handler(struct tevent_context *ev,
     struct tevent_fd *fde,
     uint16_t flags,
```c
NTSTATUS smbd_smb2_request_dispatch(struct smbd_smb2_request *req)
{
    
    DTRACE_PROBE2(smb2, request_start, mid, opcode);
    /*
     * Check if the client provided a valid session id,
     * if so smbd_smb2_request_check_session() calls
     */

    NTSTATUS smbd_smb2_request_done_ex(struct smbd_smb2_request *req,
                                        struct iovec *outbody_v;
                                        struct iovec *outdyn_v;
                                        uint32_t next_command_ofs;
                                        
                                        uint64_t mid;
                                        int16_t opcode;
                                        DEBUG(10,("smbd_smb2_request_done_ex: 
                                            "idx[%d] status[%s] body[%u] dyn[%s:%u] at %s\n",
                                        outbody_v = SMBD_SMB2_OUT_BODY_IOV(req);
                                        outdyn_v = SMBD_SMB2_OUT_DYN_IOV(req);
                                        
                                        mid = BVAL(outhdr, SMB2_HDR_MESSAGE_ID);
                                        opcode = SVAL(outhdr, SMB2_HDR_OPCODE);
                                        DTRACE_PROBE2(smb2, request_end, mid, opcode);
                                        next_command_ofs = IVAL(outhdr, SMB2_HDR_NEXT_COMMAND);
                                        SIVAL(outhdr, SMB2_HDR_STATUS, NT_STATUS_V(status));
```

2.2. SYSTEMTAP

```bash
$ grep -A 1 'DTRACE_PROBE(' /usr/include/sys/sdt.h
#define DTRACE_PROBE(provider,probe) \
    STAP_PROBE(provider,probe)
```
While at it, let’s also add instrumentation for these:

1. tevent (epoll_wait() in the epoll backend, could use syscall tracing instead)

2. sending / receiving data from the network

3. syscalls (so we don’t have to use bloody strace)

4. smbd ⇔ ctdb communication latency

Let me introduce you to tsmbd:
2.2. SYSTEMTAP

Trace an smbd process with Systemtap

USAGE: tsmbd [-s|-d|-h] pid

- pid # trace this process ID
- s # show syscall details
- d # show distribution histograms
- h # print this help text

$ sudo ./script/tsmbd -h
Trace an smbd process with Systemtap
2.2. SYSTEMTAP

$ sudo ./script/tsmbd 27376
Collecting data, press ctrl-C to stop...
2.2. SYSTEMTAP

$ sudo ./script/tsmbd 27376
Collecting data, press ctrl-C to stop... ^C

Ran for: 17147 ms

Time waiting for events: 9302 ms
Time receiving SMB2 packets: 224 ms
Time parsing SMB2 packets: 92 ms
Time running SMB2 requests: 5744 ms
Time sending SMB2 packets: 1508 ms
Time waiting for ctdb: 0 ms

Time sum: 16871 ms

Time in syscalls: 2307 ms
Time in READ disk IO: 0 ms
Time in WRITE disk IO: 167 ms

Number of tevent events: 51214
Number of SMB2 requests: 41212
Number of ctdb requests: 0

SMB2 Requests | Count | Total us | Avg us | Min us | Max us |
---------------|-------|----------|--------|--------|--------|
SMB2_OP_GETINFO | 10201 | 311821 | 30 | 15 | 603 |
SMB2_OP_CREATE | 10507 | 3137339 | 298 | 80 | 6985 |
SMB2_OP_CLOSE | 10403 | 1189089 | 114 | 23 | 2931 |
SMB2_OP_SETINFO | 101 | 3717 | 36 | 29 | 199 |
SMB2_OP_WRITE | 10000 | 1081732 | 108 | 52 | 3614 |
2.2. SYSTEMTAP

```
$ sudo ./script/tsmbd 27376
Collecting data, press ctrl-C to stop... ^C

Ran for: 17147 ms

**Time waiting for events:** 9302 ms
Time receiving SMB2 packets: 224 ms
Time parsing SMB2 packets: 92 ms
Time running SMB2 requests: 5744 ms
Time sending SMB2 packets: 1508 ms
Time waiting for ctdb: 0 ms

Time sum: 16871 ms

Time in syscalls: 2307 ms
Time in READ disk IO: 0 ms
Time in WRITE disk IO: 167 ms

Number of tevent events: 51214
Number of SMB2 requests: 41212
Number of ctdb requests: 0

<table>
<thead>
<tr>
<th>SMB2 Requests</th>
<th>Count</th>
<th>Total us</th>
<th>Avg us</th>
<th>Min us</th>
<th>Max us</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMB2_OP_GETINFO</td>
<td>10201</td>
<td>311821</td>
<td>30</td>
<td>15</td>
<td>603</td>
</tr>
<tr>
<td>SMB2_OP_CREATE</td>
<td>10507</td>
<td>3137339</td>
<td>298</td>
<td>80</td>
<td>6985</td>
</tr>
<tr>
<td>SMB2_OP_CLOSE</td>
<td>10403</td>
<td>1189089</td>
<td>114</td>
<td>23</td>
<td>2931</td>
</tr>
<tr>
<td>SMB2_OP_SETINFO</td>
<td>101</td>
<td>3717</td>
<td>36</td>
<td>29</td>
<td>199</td>
</tr>
<tr>
<td>SMB2_OP_WRITE</td>
<td>10000</td>
<td>1081732</td>
<td>108</td>
<td>52</td>
<td>3614</td>
</tr>
</tbody>
</table>
```
2.2. SYSTEMTAP

```
$ sudo ./script/tsmbd 27376
Collecting data, press ctrl-C to stop... ^C

Ran for: 17147 ms
Time waiting for events: 9302 ms
**Time receiving SMB2 packets:** 224 ms
Time parsing SMB2 packets: 92 ms
Time running SMB2 requests: 5744 ms
Time sending SMB2 packets: 1508 ms
Time waiting for ctdb: 0 ms
==========================================
Time sum: 16871 ms

Time in syscalls: 2307 ms
Time in READ disk IO: 0 ms
Time in WRITE disk IO: 167 ms

Number of tevent events: 51214
Number of SMB2 requests: 41212
Number of ctdb requests: 0

<table>
<thead>
<tr>
<th>SMB2 Requests</th>
<th>Count</th>
<th>Total us</th>
<th>Avg us</th>
<th>Min us</th>
<th>Max us</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMB2_OP_GETINFO</td>
<td>10201</td>
<td>311821</td>
<td>30</td>
<td>15</td>
<td>603</td>
</tr>
<tr>
<td>SMB2_OP_CREATE</td>
<td>10507</td>
<td>3137339</td>
<td>298</td>
<td>80</td>
<td>6985</td>
</tr>
<tr>
<td>SMB2_OP_CLOSE</td>
<td>10403</td>
<td>1189089</td>
<td>114</td>
<td>23</td>
<td>2931</td>
</tr>
<tr>
<td>SMB2_OP_SETINFO</td>
<td>101</td>
<td>3717</td>
<td>36</td>
<td>29</td>
<td>199</td>
</tr>
<tr>
<td>SMB2_OP_WRITE</td>
<td>10000</td>
<td>1081732</td>
<td>108</td>
<td>52</td>
<td>3614</td>
</tr>
</tbody>
</table>
```
2.2. SYSTEMTAP

```bash
$ sudo ./script/tsmbd 27376
Collecting data, press ctrl-C to stop... ^C

Ran for:                          17147 ms
Time waiting for events:           9302 ms
Time receiving SMB2 packets:        224 ms
Time parsing SMB2 packets:          92 ms
Time running SMB2 requests:         5744 ms
Time sending SMB2 packets:          1508 ms
**Time waiting for ctdb:**           0 ms
==========================================
Time sum:                         16871 ms
Time in syscalls:                  2307 ms
Time in READ disk IO:              0 ms
Time in WRITE disk IO:             167 ms

Number of tevent events:          51214
Number of SMB2 requests:          41212
Number of ctdb requests:           0

SMB2 Requests                     Count  Total us  Avg us  Min us  Max us
SMB2_OP_GETINFO                   10201  311821   30      15     603
SMB2_OP_CREATE                    10507  3137339  298     80    6985
SMB2_OP_CLOSE                     10403  1189089  114     23    2931
SMB2_OP_SETINFO                   101    3717     36      29     199
SMB2_OP_WRITE                     10000  1081732  108     52    3614
```
2.2. SYSTEMTAP

```
$ sudo ./script/tsmbd 27376
Collecting data, press ctrl-C to stop... ^C

Ran for:                          17147 ms
Time waiting for events:           9302 ms
Time receiving SMB2 packets:        224 ms
Time parsing SMB2 packets:          92 ms
Time running SMB2 requests:         5744 ms
Time sending SMB2 packets:          1508 ms
Time waiting for ctdb:               0 ms
Time sum:                          16871 ms
Time in syscalls:                   2307 ms
Time in READ disk IO:               0 ms
Time in WRITE disk IO:              167 ms
Number of tevent events:           51214
Number of SMB2 requests:           41212
Number of ctdb requests:            0

SMB2 Requests                  Count     Total us       Avg us       Min us       Max us
SMB2_OP_GETINFO                10201       311821           30          15            603
SMB2_OP_CREATE                 10507      3137339          298          80           6985
SMB2_OP_CLOSE                  10403      1189089          114          23           2931
SMB2_OP_SETINFO                  101        3717           36           29            199
SMB2_OP_WRITE                  10000      1081732          108          52           3614
```
2.2. SYSTEMTAP

```bash
$ sudo ./script/tsmbd 27376
Collecting data, press ctrl-C to stop... ^C

Ran for:                          17147 ms
Time waiting for events:           9302 ms
Time receiving SMB2 packets:        224 ms
Time parsing SMB2 packets:          92 ms
Time running SMB2 requests:         5744 ms
Time sending SMB2 packets:          1508 ms
Time waiting for ctdb:               0 ms
==========================================
Time sum:                         16871 ms
Time in syscalls:                  2307 ms
Time in READ disk IO:              0 ms
Time in WRITE disk IO:             167 ms

Number of tevent events:          51214
Number of SMB2 requests:          41212
Number of ctdb requests:           0

SMB2 Requests                  Count     Total us       Avg us       Min us       Max us
SMB2_OP_GETINFO                10201       311821           30          15         603
SMB2_OP_CREATE                 10507      3137339          298          80        6985
SMB2_OP_CLOSE                  10403      1189089          114          23        2931
SMB2_OP_SETINFO                  101         3717           36          29         199
SMB2_OP_WRITE                  10000      1081732          108          52        3614
```
2.2. SYSTEMTAP

$ sudo ./script/tsmbd 27376
Collecting data, press ctrl-C to stop... ^C

Ran for: 17147 ms

Time waiting for events: 9302 ms
Time receiving SMB2 packets: 224 ms
Time parsing SMB2 packets: 92 ms
Time running SMB2 requests: 5744 ms
Time sending SMB2 packets: 1508 ms
Time waiting for ctdb: 0 ms

Time sum: 16871 ms

Time in syscalls: 2307 ms
Time in READ disk IO: 0 ms
Time in WRITE disk IO: 167 ms

Number of tevent events: 51214
Number of SMB2 events: 41212
Number of ctdb requests: 0

<table>
<thead>
<tr>
<th>SMB2 Requests</th>
<th>Count</th>
<th>Total us</th>
<th>Avg us</th>
<th>Min us</th>
<th>Max us</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMB2_OP_GETINFO</td>
<td>10201</td>
<td>311821</td>
<td>30</td>
<td>15</td>
<td>603</td>
</tr>
<tr>
<td>SMB2_OP_CREATE</td>
<td>10507</td>
<td>3137339</td>
<td>298</td>
<td>80</td>
<td>6985</td>
</tr>
<tr>
<td>SMB2_OP_CLOSE</td>
<td>10403</td>
<td>1189089</td>
<td>114</td>
<td>23</td>
<td>2931</td>
</tr>
<tr>
<td>SMB2_OP_SETINFO</td>
<td>101</td>
<td>3717</td>
<td>36</td>
<td>29</td>
<td>199</td>
</tr>
<tr>
<td>SMB2_OP_WRITE</td>
<td>10000</td>
<td>1081732</td>
<td>108</td>
<td>52</td>
<td>3614</td>
</tr>
</tbody>
</table>
$ sudo ./script/tsmbd 27376
Collecting data, press ctrl-C to stop... ^C

Ran for:                          17147 ms
Time waiting for events:           9302 ms
Time receiving SMB2 packets:        224 ms
Time parsing SMB2 packets:          92 ms
Time running SMB2 requests:         5744 ms
Time sending SMB2 packets:          1508 ms
Time waiting for ctdb:              0 ms
==========================================
Time sum:                         16871 ms

Time in syscalls:                  2307 ms
Time in READ disk IO:               0 ms
Time in WRITE disk IO:              167 ms

Number of tevent events:           51214
Number of SMB2 requests:           41212
Number of ctdb requests:            0

<table>
<thead>
<tr>
<th>SMB2 Requests</th>
<th>Count</th>
<th>Total us</th>
<th>Avg us</th>
<th>Min us</th>
<th>Max us</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMB2_OP_GETINFO</td>
<td>10201</td>
<td>311821</td>
<td>30</td>
<td>15</td>
<td>603</td>
</tr>
<tr>
<td>SMB2_OP_CREATE</td>
<td>10507</td>
<td>3137339</td>
<td>298</td>
<td>80</td>
<td>6985</td>
</tr>
<tr>
<td>SMB2_OP_CLOSE</td>
<td>10403</td>
<td>1189089</td>
<td>114</td>
<td>23</td>
<td>2931</td>
</tr>
<tr>
<td>SMB2_OP_SETINFO</td>
<td>101</td>
<td>3717</td>
<td>36</td>
<td>29</td>
<td>199</td>
</tr>
<tr>
<td>SMB2_OP_WRITE</td>
<td>10000</td>
<td>1081732</td>
<td>108</td>
<td>52</td>
<td>3614</td>
</tr>
</tbody>
</table>
2.2. SYSTE MTAP

```bash
$ sudo ./script/tsmbd 27376
Collecting data, press ctrl-C to stop... ^C

Ran for: 17147 ms
Time waiting for events: 9302 ms
Time receiving SMB2 packets: 224 ms
Time parsing SMB2 packets: 92 ms
Time running SMB2 requests: 5744 ms
Time sending SMB2 packets: 1508 ms
Time waiting for ctdb: 0 ms

Time sum: 16871 ms

Time in syscalls: 2307 ms
Time in READ disk IO: 0 ms
Time in WRITE disk IO: 167 ms

Number of tevent events: 51214
Number of SMB2 requests: 41212
Number of ctdb requests: 0

<table>
<thead>
<tr>
<th>SMB2 Requests</th>
<th>Count</th>
<th>Total us</th>
<th>Avg us</th>
<th>Min us</th>
<th>Max us</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMB2_OP_GETINFO</td>
<td>10201</td>
<td>311821</td>
<td>30</td>
<td>15</td>
<td>603</td>
</tr>
<tr>
<td>SMB2_OP_CREATE</td>
<td>10507</td>
<td>3137339</td>
<td>298</td>
<td>80</td>
<td>6985</td>
</tr>
<tr>
<td>SMB2_OP_CLOSE</td>
<td>10403</td>
<td>1189089</td>
<td>114</td>
<td>23</td>
<td>2931</td>
</tr>
<tr>
<td>SMB2_OP_SETINFO</td>
<td>101</td>
<td>3717</td>
<td>36</td>
<td>29</td>
<td>199</td>
</tr>
<tr>
<td>SMB2_OP_WRITE</td>
<td>10000</td>
<td>1081732</td>
<td>108</td>
<td>52</td>
<td>3614</td>
</tr>
</tbody>
</table>
```
<table>
<thead>
<tr>
<th>System call</th>
<th>Count</th>
<th>Total us</th>
</tr>
</thead>
<tbody>
<tr>
<td>geteuid</td>
<td>16</td>
<td>7</td>
</tr>
<tr>
<td>getegid</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>setgroups</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>setresgid</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>setresuid</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>stat</td>
<td>215373</td>
<td>256907</td>
</tr>
<tr>
<td>getcwd</td>
<td>50989</td>
<td>26160</td>
</tr>
<tr>
<td>open</td>
<td>20205</td>
<td>312274</td>
</tr>
<tr>
<td>fstat</td>
<td>50206</td>
<td>45133</td>
</tr>
<tr>
<td>getdents</td>
<td>20410</td>
<td>175707</td>
</tr>
<tr>
<td>close</td>
<td>20207</td>
<td>15967</td>
</tr>
<tr>
<td>lstat</td>
<td>61208</td>
<td>52244</td>
</tr>
<tr>
<td>read</td>
<td>42021</td>
<td>58799</td>
</tr>
<tr>
<td>fcntl</td>
<td>470178</td>
<td>408604</td>
</tr>
<tr>
<td>getxattr</td>
<td>62719</td>
<td>201745</td>
</tr>
<tr>
<td>mkdir</td>
<td>101</td>
<td>4191</td>
</tr>
<tr>
<td>setxattr</td>
<td>10101</td>
<td>136778</td>
</tr>
<tr>
<td>getgroups</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>sendmsg</td>
<td>30202</td>
<td>298217</td>
</tr>
<tr>
<td>listxattr</td>
<td>201</td>
<td>528</td>
</tr>
<tr>
<td>chdir</td>
<td>40400</td>
<td>38939</td>
</tr>
<tr>
<td>fgetxattr</td>
<td>10000</td>
<td>10658</td>
</tr>
<tr>
<td>clock_gettime</td>
<td>20001</td>
<td>17726</td>
</tr>
<tr>
<td>pwrite</td>
<td>10014</td>
<td>159469</td>
</tr>
<tr>
<td>write</td>
<td>10000</td>
<td>40025</td>
</tr>
<tr>
<td>utimensat</td>
<td>10000</td>
<td>47363</td>
</tr>
</tbody>
</table>
### 2.2. SYSTEMTAP

#### SMB2_OP_CREATE distribution (microseconds)

<table>
<thead>
<tr>
<th>Value</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>32</td>
<td>0</td>
</tr>
<tr>
<td>64</td>
<td>289</td>
</tr>
<tr>
<td>128</td>
<td>846</td>
</tr>
<tr>
<td>256</td>
<td>9315</td>
</tr>
<tr>
<td>512</td>
<td>48</td>
</tr>
<tr>
<td>1024</td>
<td>7</td>
</tr>
<tr>
<td>2048</td>
<td>0</td>
</tr>
<tr>
<td>4096</td>
<td>1</td>
</tr>
<tr>
<td>8192</td>
<td>2</td>
</tr>
<tr>
<td>16384</td>
<td>0</td>
</tr>
<tr>
<td>32768</td>
<td>0</td>
</tr>
</tbody>
</table>

#### WRITE IO distribution (microseconds)

<table>
<thead>
<tr>
<th>Value</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>5913</td>
</tr>
<tr>
<td>16</td>
<td>3909</td>
</tr>
<tr>
<td>32</td>
<td>149</td>
</tr>
<tr>
<td>64</td>
<td>27</td>
</tr>
<tr>
<td>128</td>
<td>0</td>
</tr>
<tr>
<td>256</td>
<td>0</td>
</tr>
<tr>
<td>512</td>
<td>1</td>
</tr>
<tr>
<td>1024</td>
<td>0</td>
</tr>
<tr>
<td>2048</td>
<td>1</td>
</tr>
<tr>
<td>4096</td>
<td>0</td>
</tr>
<tr>
<td>8192</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Systemtap summary:

- Low overhead, not completely free, but cheap
  - not traced: 13 s
  - traced: 14 s
- It would be easy to enhance tsmbd to trace all smbd (as an option)
- But this may turn out to be not so cheap anymore :-(
Because:

1. 20% network IO and SMB2 packet processing
2. 40% waiting for requests
3. 40% CPU usage for SMB2 fileserver engine
LINUX
PERF

perf_events
Remember 50% CPU over 13 seconds

compared to ~100% over 0.3 seconds for local copy

So a tremendous computation overhead, smbd is doing a *lot*

But what is it doing?

tsmbd told us 30% CPU is in the kernel servicing syscalls

the rest is user-space CPU time (20%)

So let’s eliminate syscalls and optimise our code

But where? We need profiling info
Linux profilers:

1. GNU gprof: requires special compilation
2. Valgrind Callgrind: slooooooooooow
3. oprofile
4. …or…
perf: a kernel subsystem(s) and a user-space tool

it can instrument CPU performance counters, tracepoints, kprobes, and uprobes (go read up on it on the web, I’m not going to explain that here)

Put it differently: like Systemtap without the scripts

It aggregates metrics and dumps them in a file:
# perf record [-p pid]

You then use the user-space tool perf to print the data
# perf report[-p pid]

Can sample the stack of processes, including kernel stack
# perf record -g
It has an awesome interface to display the stack profile info:

```
# perf report -g ...
```

Example:
### 2.3. PERF

**Samples:** 2K of event 'cpu-clock', Event count (approx.): 685500000

<table>
<thead>
<tr>
<th>Children</th>
<th>Self</th>
<th>Command</th>
<th>Shared Object</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>52.48%</td>
<td>0.15%</td>
<td>smbd</td>
<td>libtevent.so</td>
</tr>
<tr>
<td></td>
<td>52.44%</td>
<td>0.00%</td>
<td>smbd</td>
<td>libsmbd-base-samba4.so</td>
</tr>
<tr>
<td></td>
<td>52.44%</td>
<td>0.22%</td>
<td>libtevent.so</td>
<td>smbd_process</td>
</tr>
<tr>
<td></td>
<td>52.44%</td>
<td>0.18%</td>
<td>libtevent.so</td>
<td>smbd_accept_connection</td>
</tr>
<tr>
<td></td>
<td>52.44%</td>
<td>0.22%</td>
<td>libtevent.so</td>
<td>std_event_loop_once</td>
</tr>
<tr>
<td></td>
<td>52.44%</td>
<td>0.00%</td>
<td>libtevent.so</td>
<td>_tevent_loop_once</td>
</tr>
<tr>
<td></td>
<td>52.44%</td>
<td>0.00%</td>
<td>libtevent.so</td>
<td>std_event_loop_wait</td>
</tr>
<tr>
<td></td>
<td>52.44%</td>
<td>0.00%</td>
<td>libtevent.so</td>
<td>_tevent_loop_wait</td>
</tr>
<tr>
<td></td>
<td>52.44%</td>
<td>0.00%</td>
<td>libtevent.so</td>
<td>_tevent_loop_wait</td>
</tr>
<tr>
<td></td>
<td>52.44%</td>
<td>0.22%</td>
<td>libtevent.so</td>
<td>smbd_smb2_create_send</td>
</tr>
<tr>
<td></td>
<td>49.12%</td>
<td>0.66%</td>
<td>smbd</td>
<td>[kernel.vmlinux] entry_SYSCALL_64_fastpath</td>
</tr>
<tr>
<td></td>
<td>33.95%</td>
<td>0.00%</td>
<td>smbd</td>
<td>libsmbd-base-samba4.so smb_smb2_connection_handler</td>
</tr>
<tr>
<td></td>
<td>33.84%</td>
<td>0.26%</td>
<td>libsmbd-base-samba4.so smb_smb2_io_handler</td>
<td></td>
</tr>
<tr>
<td></td>
<td>32.79%</td>
<td>0.15%</td>
<td>libsmbd-base-samba4.so smb_smb2_request_dispatch</td>
<td></td>
</tr>
<tr>
<td></td>
<td>20.75%</td>
<td>0.00%</td>
<td>libsmbd-base-samba4.so smb_smb2_request_process_create</td>
<td></td>
</tr>
<tr>
<td></td>
<td>18.23%</td>
<td>0.07%</td>
<td>libsmbd-base-samba4.so smb_smb2_create_send</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12.84%</td>
<td>0.07%</td>
<td>smbd</td>
<td>[kernel.vmlinux] do_readv_writev</td>
</tr>
<tr>
<td></td>
<td>12.80%</td>
<td>0.00%</td>
<td>libsmbd-base-samba4.so create_file_default</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12.80%</td>
<td>0.00%</td>
<td>libsmbd-base-samba4.so vfswrap_create_file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12.80%</td>
<td>0.00%</td>
<td>libsmbd-base-samba4.so smb_vfs_call_create_file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12.76%</td>
<td>0.04%</td>
<td>libsmbd-base-samba4.so create_file_unixpath</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12.47%</td>
<td>0.00%</td>
<td>[kernel.vmlinux] sock_sendmsg</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12.33%</td>
<td>0.07%</td>
<td>[kernel.vmlinux] do_iter_readv_writev</td>
<td></td>
</tr>
<tr>
<td></td>
<td>11.82%</td>
<td>11.82%</td>
<td>smbd</td>
<td>[kernel.vmlinux] _raw_spin_unlock_irqrestore</td>
</tr>
<tr>
<td></td>
<td>11.71%</td>
<td>0.00%</td>
<td>[kernel.vmlinux] do_writev</td>
<td></td>
</tr>
<tr>
<td></td>
<td>11.71%</td>
<td>0.00%</td>
<td>[kernel.vmlinux] sys_writev</td>
<td></td>
</tr>
<tr>
<td></td>
<td>11.67%</td>
<td>0.00%</td>
<td>[kernel.vmlinux] vfs_writev</td>
<td></td>
</tr>
<tr>
<td></td>
<td>11.56%</td>
<td>0.00%</td>
<td>[kernel.vmlinux] sock_write_iter</td>
<td></td>
</tr>
</tbody>
</table>
There's a nice visualization for call-stack profiles: Flamegraphs
2.3. PERF

- Try them out, they are a good starting point
- More on Flamegraphs hopefully in the next talk from Douglas and later by Andrew
PERF DEMO
SMBD PERFORMANCE IMPROVEMENTS
3. SMBD PERFORMANCE IMPROVEMENTS

1. Clustered Samba: directory enumeration
2. Name mangling: new option „mangled names = illegal“
3. Make use of struct smb_filename plumbing in the 4.5 VFS
4. GPFS VFS module improvements
5. Messaging
Clustered Samba: directory enumeration improvements

- enumerating a directory requires asking locking.tdb for every directory entry (we want the file write time from a possible open file record)
- on a cluster this goes via ctdb and takes some time
- smbd sits idle waiting for the answer
- the fix: add an async dwrap_fetch_send/recv API
- uses async communication with ctdb at the low level
- smbd can now pipeline the requests and continue processing the directory
- at the end it collects the results
- Time for enumerating directory with 50k file goes from 10 s to 3 s
3. NAME MANGLING = ILLEGAL

- man smb.conf: „mangled names = … controls whether non-DOS names under UNIX should be mapped to DOS-compatible names …“
- any name longer then 8.3 is a non-DOS name
- this means that with SMB2+ we’re still providing mangled names in FIND responses, Windows server doesn't
- calculating the mangled name mapping is awfully expensive
- new option setting: „mangled names = illegal“
- only add mapping for filenames with illegal NTFS characters that still really require mangling
as part of the CREATE call we allocate a struct smb_filename

for existing files „struct smb_filename->SMB_STRUCT_STAT st“ will already contain valid stat info

in Samba 4.4 many VFS functions weren’t passed the smb_filename but a char * with just the path

so sometimes VFS modules and parts of smbd had to stat again

thanks to Jeremy struct smb_filename is now plumbed through the VFS

so I could remove a few stats in a few places
stat functions in vfs_gpfs were all calling into GPFS to update the file creation time

stat may be called multiple times from different contexts, every time we call down into GPFS

but at the top level smbd uses a different VFS call (SMB_VFS_GET_DOS_ATTRIBUTES) to request file creation time from the VFS

so after adding the creation time hook to the GPFS SMB_VFS_GET_DOS_ATTRIBUTES implementation, remove the calls into GPFS from the stat hooks
3. GPFS VFS MODULE / NFSV4 ACL PARMS

- Initialize smb.conf NFSv4 ACL parameters at tree connect (from Volker)

- Without this we call out to smb.conf parsing code for every ACL related VFS function in VFS GPFS which is bad, bad, bad
for every message smbd processes send to one another we go through the full machinery of connection setup, send message and connection teardown

but there are connections that could be reused: file change notifications from smbd processes to notifyd

solution: keep idle connections for 1 s

this helps a lot for the small file copy workload
3. RESULT

Small file copy throughput:

- before: 136 files / s
- after: 151 files / s
- ~10% improvement just by optimising existing code, no new code
4. LINKS

https://git.samba.org/?p=slow/samba.git;a=shortlog;h=refs/heads/perf
THANK YOU!

QUESTIONS?

Ralph Böhme <slow@samba.org>