Profiling Linux Operations for Performance and Troubleshooting

by Tanel Põder

https://tanelpoder.com/

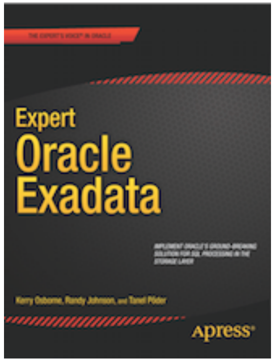
@tanelpoder

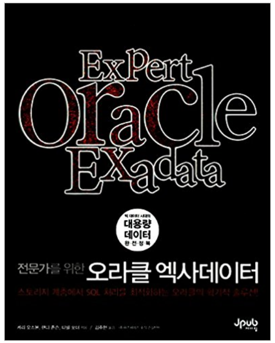
About me

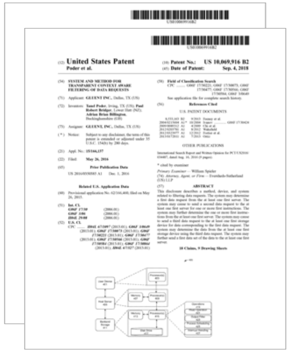
- **Tanel Põder**
  - I'm a database performance geek (23 years)
  - Before that an Unix/Linux geek, (27 years)
  - Oracle, Hadoop, Spark, cloud databases ☺
  - Focused on performance & troubleshooting

  - **Inventing & hacking stuff, consulting, training**
  - Co-author of the Expert Oracle Exadata book
  - Co-founder & technical advisor at Gluent
  - 2 patents in data virtualization space
  - Working on a secret project ;-)

Blog:	tanelpoder.com
Twitter:	twitter.com/TanelPoder
Questions:	tanel@tanelpoder.com






(12) United States Patent
Poder et al.

(10) Patent No.: US 10,069,916 B2
(45) Date of Patent: Sep. 4, 2018

(54) SYSTEM AND METHOD FOR TRANSPARENT CONTEXT AWARE FILTERING OF DATA REQUESTS

(71) Applicant: GLUENT INC., Dallas, TX (US)

(72) Inventors: Tanel Poder, Irving, TX (US); Paul Robert Bridger, Lower Hutt (NZ); Adrian Brian Billington, Buckinghamshire (GB)

(73) Assignee: GLUENT, INC., Dallas, TX (US)

PODERC

ORACLE | CERTIFIED MASTER

gluent.

alumni

OakTable.net

Agenda

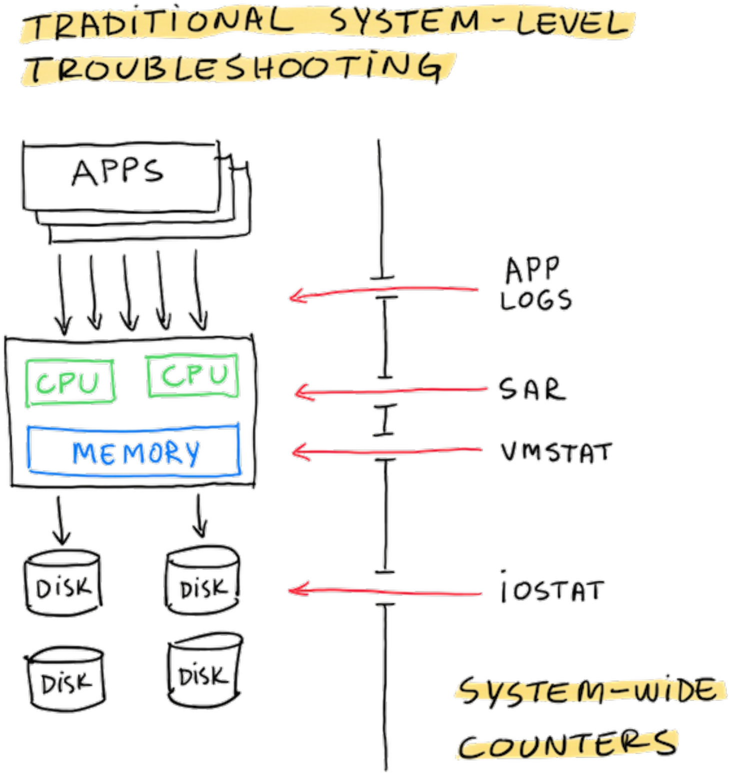
1. A *short* intro to Linux **task state sampling** method
2. Demos
3. More Demos
4. Always on profiling of production systems

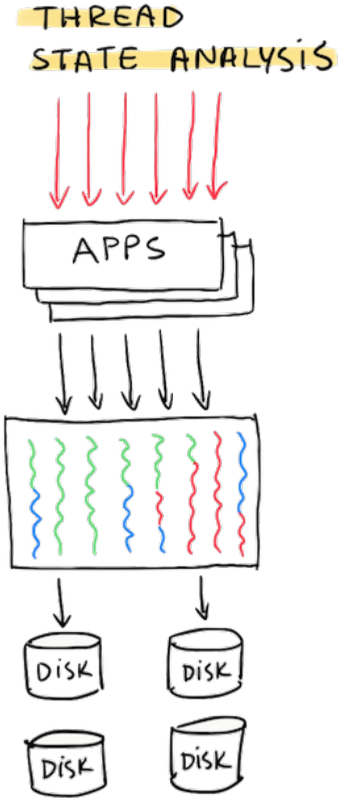
Preferring low-tech tools for high-tech problems

- Why?
  - I do ad-hoc troubleshooting for different customers
  - No time to *engineer* a solution, the problem is already happening
  - Troubleshooting across a variety of servers, distros, installations
  - Old Linux distro/kernel versions
  - No permission to change anything (including enabling kernel tracing)
  - Sometimes no root access
  - Idea: Ultra-low footprint tools that get the most out of already enabled Linux instrumentation
    - **/proc** filesystem!

Low tech tools aren't always "deep" enough or precise enough, but they are quick & easy to try out

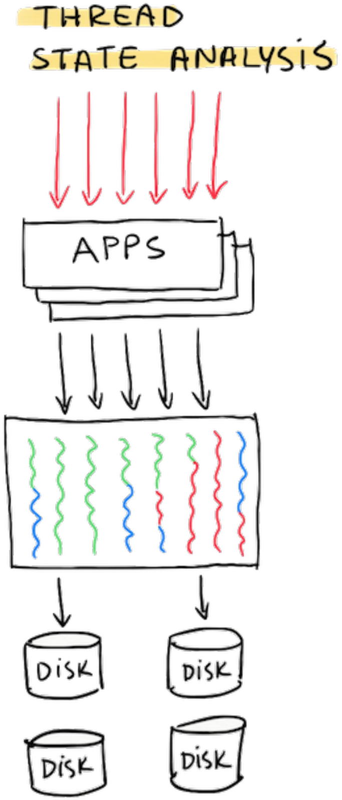
System-level metrics & thread state analysis





Let's sample the threads!

Application thread state analysis tools



- Classic Linux tools
  - ps
  - top -> (htop, atop, nmon, …)

These tools also sample, snapshot /proc files

- Custom /proc sampling tools
  - **0x.tools pSnapper**
  - **0x.tools xcapture**
  - grep . /proc/*/stat

Proc sampling complements, not replaces other tools

- Linux (kernel) tracing tools
  - perf top, perf record, perf probe
  - strace
  - SystemTap, eBPF/bpftrace
- Application level tools
  - JVM attach + profile
  - Python attach + profile

Listing processes & threads

```
$ ps -o pid,ppid,tid,thcount,comm -p 1994
  PID  PPID   TID THCNT COMMAND
 1994  1883  1994   157 java
```

Multi-threaded JVM process

```
$ ps -o pid,ppid,tid,thcount,comm -L -p 1994 | head
  PID  PPID   TID THCNT COMMAND
 1994  1883  1994   157 java       <-- thread group leader
 1994  1883  2008   157 java
 1994  1883  2011   157 java
 1994  1883  2014   157 java
...
```

List each thread individually

Thread group leader thread PID == TID

```
$ ps -eLf | wc -l
1162

$ ls -ld /proc/[0-9]* | wc -l
804

$ ls -ld /proc/[0-9]*/task/* | wc -l
1161
```

All threads are visible in /proc

Non-leader threads are listed in *task* subdirectories

Task states

- Every thread (task) has a *"current state"* flag
  - Updated by kernel functions just before they call schedule()
  - Visible in `/proc/PID/stat` & `/proc/PID/status`

```
$ man ps

TASK STATES

D    uninterruptible sleep (usually IO)
R    running or runnable (on run queue)
S    interruptible sleep (waiting for an event to complete)
T    stopped by job control signal
t    stopped by debugger during the tracing
W    paging (not valid since the 2.6.xx kernel)
X    dead (should never be seen)
Z    defunct ("zombie") process, terminated but not reaped by its parent
```

*usually, not always **

R = Run**ning** + Run**nable**

Runnable = waiting for scheduler, ready to run on CPU runqueue

** We'll talk about the D state soon*

Task states - examples

- **ps -o s** reads state from `/proc/PID/stat`

```
$ ps -eo s,comm | sort | uniq -c | sort -nbr | head
     27 S sshd
     15 S bash
     15 I bioset
     13 I kdmflush
      8 S postmaster
      8 S nfsd
      8 I xfs-reclaim/dm-
      8 I xfs-eofblocks/d
      6 S httpd
      4 S sleep
```

```
$ ps -Leo s,comm,wchan | sort | uniq -c | sort -nbr | head
    152 S java            -
     72 S containerd      -
     71 S dockerd         -
     46 S java            futex_wait_queue_me
     29 S mysqld          -
     27 S sshd            -
     17 S libvirtd        -
     15 I bioset          -
     13 I kdmflush        -
     10 R mysqld          -
```

```
$ ps -eo s | sort | uniq -c | sort -nbr
    486 S
    352 I
      2 Z
      1 R
```

L – see all threads!

"s" is an alias for "state"

Show only R & D states

```
$ ps -eLo state,user,comm | grep "^[RD]" \
          | sort | uniq -c | sort -nbr
     64 R tanel    java
     24 D tanel    java
     13 R mysql    mysqld
      2 R tanel    sysbench
      2 D mysql    mysqld
      1 R tanel    ps
      1 R oracle   java
```

Task state sampling vs. vmstat

```
$ nice stress -c 32
stress: info: [28802] dispatching hogs: 32 cpu, 0 io, 0 vm, 0 hdd
```

Basic stress test tool

```
$ ps -eo state,user,comm | grep "^R" | uniq -c | sort -nbr
     32 R tanel    stress
      1 R tanel    ps
$ ps -eo state,user,comm | grep "^R" | uniq -c | sort -nbr
     32 R tanel    stress
      1 R tanel    ps
      1 R tanel    grep
```

Measurement effect: Should ignore my own "ps" and "grep" monitoring commands

```
$ vmstat 3
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
35  0 162560 26177012    276 61798720    0    0    67    45    2    0  1  0 98  0  0
32  0 162560 26177112    276 61798724    0    0    53    56 32266 1218 100  0  0  0  0
32  0 162560 26177484    276 61798724    0    0    21    13 32276 1203 100  0  0  0  0
```

vmstat "runnable" column agrees

```
$ dstat -vr
---procs--- ------memory-usage----- ---paging-- -dsk/total- ---system-- ----total-cpu-usage---- --io/total-
run blk new| used  buff  cach  free|  in   out | read  writ| int   csw |usr sys idl wai hiq siq| read  writ
0.0   0  10| 105G  276k 57.9G 25.0G|  32B  462B|  46M 2895k|2002  6740 |  1   0  98   0   0   0| 282   116
 33   0 0.7| 105G  276k 57.9G 25.0G|   0     0 |  85k   67k|  32k 1256 |100   0   0   0   0   0|5.33  3.67
 33   0  21| 105G  276k 57.9G 25.0G|   0     0 |  93k  524k|  32k 1716 |100   0   0   0   0   0|7.33  48.0
 32   0 1.0| 105G  276k 57.9G 25.0G|   0     0 |   0     0 |  32k 1235 |100   0   0   0   0   0|   0     0
```

Scheduler off-CPU reasons

- Scheduler reasons for taking threads off CPU:

	Thread State
• System CPU shortage, **R**unnable thread out of time-slice/credit • Or a higher priority process runnable	R
• Blocking I/O: within a system call (disk I/O, NFS RPC reply, lock wait) • Blocking I/O: without a system call (hard page fault)	D
• Blocking I/O: syscall against a pipe, network socket, io_getevents • Voluntary sleep: nanosleep, semtimedop, lock get	S
• Suspended with: kill -STOP, -TSTP signal • Suspended with: ptrace() by another process	T, t

- Other:
  - Linux Audit backlog, etc…

Task state *Disk sleep – uninterruptible* is not only for disk waits!

```
kernel/locking/rwsem-spinlock.c

/*
 * get a read lock on the semaphore
 */

void __sched __down_read(struct rw_semaphore *sem)
{
    struct rwsem_waiter waiter;
    struct task_struct *tsk;

    spin_lock_irq(&sem->wait_lock);

    if (sem->activity >= 0 && list_empty(&sem->wait_list)) {
        /* granted */
        sem->activity++;
        spin_unlock_irq(&sem->wait_lock);
        goto out;
    }

    tsk = current;
    set_task_state(tsk, TASK_UNINTERRUPTIBLE);
```

```
 /* set up my own style of waitqueue */
    waiter.task = tsk;
    waiter.flags = RWSEM_WAITING_FOR_READ;
    get_task_struct(tsk);

    list_add_tail(&waiter.list, &sem->wait_list);

/* we don't need to touch the semaphore struct anymore */
    spin_unlock_irq(&sem->wait_lock);

    /* wait to be given the lock */
    for (;;) {
        if (!waiter.task)
            break;
        schedule();
        set_task_state(tsk, TASK_UNINTERRUPTIBLE);
    }

    tsk->state = TASK_RUNNING;
 out:
    ;
}
```

schedule() may take task off-CPU

https://tanelpoder.com/posts/high-system-load-low-cpu-utilization-on-linux/

Threads waiting for kernel rw-spinlocks will show up with state "**D - disk wait**" !!!

Demos

0x.tools Linux Process Snapper

- A free, open source /proc file system sampling tool
  - Current: Thread state sampling (currently available)
  - Planned: Kernel counter snapshotting & deltas (CPU, IO, memory, scheduling latency etc)
  - Planned: Application profiling frontend

  - https://tanelpoder.com/psnapper

- Implementation
  - Python script (currently Python 2.6+)
  - Works with 2.6.18+ kernels (maybe older too)
  - Passive profiling - reads **/proc** files
  - Does not require installation
  - Basic usage does not require root access
    - Especially if sampling processes under your username
  - Some usage requires root access on newer kernels (wchan, kstack)

Linux Process Snapper

- More info:
  - psn -h
  - psn --list
  - https://0x.tools

```
$ psn

Process Snapper sampling cmdline, stat for 5 seconds...
finished sampling

=== Active Threads ==============================================

 samples | avg_threads | cmdline          | state
-----------------------------------------------------------------
     316 |        9.58 | fio              | Disk (Uninterruptible)
     212 |        6.42 | fio              | Running (ON CPU)
      33 |        1.00 | python           | Running (ON CPU)
      30 |        0.91 |                  | Running (ON CPU)
       3 |        0.09 |                  | Disk (Uninterruptible)
       2 |        0.06 | /usr/bin/perl    | Running (ON CPU)
       1 |        0.03 | ora_vktm_LINPRD  | Running (ON CPU)
       1 |        0.03 | top              | Running (ON CPU)
```

```
$ psn -p 18286 -G syscall,filename

Linux Process Snapper v0.14 by Tanel Poder [https://tp.dev/psnapper]
Sampling /proc/stat, syscall for 5 seconds... finished.


=== Active Threads ==================================================================================

 samples | avg_threads | comm | state                  | syscall   | filename
-----------------------------------------------------------------------------------------------------
      79 |        0.79 | (dd) | Disk (Uninterruptible) | write     | /backup/tanel/test (stdout)
       7 |        0.07 | (dd) | Disk (Uninterruptible) | [running] |
       5 |        0.05 | (dd) | Running (ON CPU)       | write     | /backup/tanel/test (stdout)
       4 |        0.04 | (dd) | Disk (Uninterruptible) | read      | /reco/fio/mmapfile.0.0 (stdin)
       3 |        0.03 | (dd) | Running (ON CPU)       | [running] |
       2 |        0.02 | (dd) | Running (ON CPU)       | read      | /reco/fio/mmapfile.0.0 (stdin)
```

Linux Process Snapper

```
$ sudo psn -G syscall,wchan -r -p "sync|kworker"

Linux Process Snapper v0.11 by Tanel Poder [https://tp.dev/psnapper]
Sampling /proc/stat, syscall, wchan for 5 seconds... finished.


=== Active Threads =====================================================================================

 samples | avg_threads | comm            | state                  | syscall    | wchan
--------------------------------------------------------------------------------------------------------
     100 |        1.00 | (sync)          | Disk (Uninterruptible) | sync       | wb_wait_for_completion
      98 |        0.98 | (kworker/u66:0) | Disk (Uninterruptible) | read       | wait_barrier
      82 |        0.82 | (md10_resync)   | Disk (Uninterruptible) | read       | raise_barrier
      15 |        0.15 | (md10_resync)   | Disk (Uninterruptible) | read       | md_do_sync
       3 |        0.03 | (kworker/29:2)  | Disk (Uninterruptible) | read       | rpm_resume
       3 |        0.03 | (md10_resync)   | Disk (Uninterruptible) | read       | raid10_sync_request
       2 |        0.02 | (kworker/1:0)   | Disk (Uninterruptible) | read       | hub_event
       2 |        0.02 | (kworker/29:2)  | Disk (Uninterruptible) | read       | msleep
       1 |        0.01 | (kworker/20:1H) | Running (ON CPU)       | read       | worker_thread
       1 |        0.01 | (kworker/30:0)  | Running (ON CPU)       | [userland] | 0
       1 |        0.01 | (kworker/6:0)   | Running (ON CPU)       | [userland] | 0
       1 |        0.01 | (kworker/u66:0) | Running (ON CPU)       | [userland] | 0
       1 |        0.01 | (kworker/u66:0) | Running (ON CPU)       | read       | wait_barrier
```

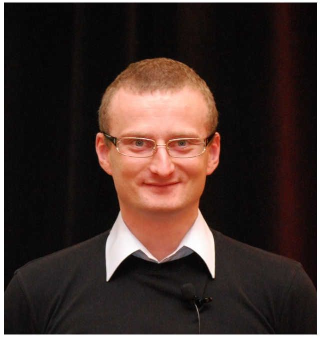
Always-on profiling of production systems?

- **0x.tools**
  - https://0x.tools
  - https://twitter.com/0xtools
  - Open Source (GPLv3)
  - Low-footprint & low-overhead (no large dependencies)

  - **xcapture** – samples **/proc** states like pSnapper
  - **run_xcpu.sh** – uses **perf** for on-CPU stack sampling at 1 Hz

- Always-on low-frequency sampling of on-CPU & thread sleep samples
  - xcapture outputs hourly .csv files ("query" with anything)
  - perf logs can be used just with perf report -i xcpu.20201201100000

Thank you!



Tanel Põder

A long time computer performance geek

- Blog, Tools, Videos, Articles
  - https://tanelpoder.com/categories/linux
  - https://tanelpoder.com/videos
  - https://0x.tools
- Events, Hacking Sessions, Online Training
  - https://tanelpoder.com/events/
- Contact
  - Blog: tanelpoder.com
  - Twitter: twitter.com/TanelPoder
  - Questions: tanel@tanelpoder.com