
# Improving performance of Mission Critical applications on Linux

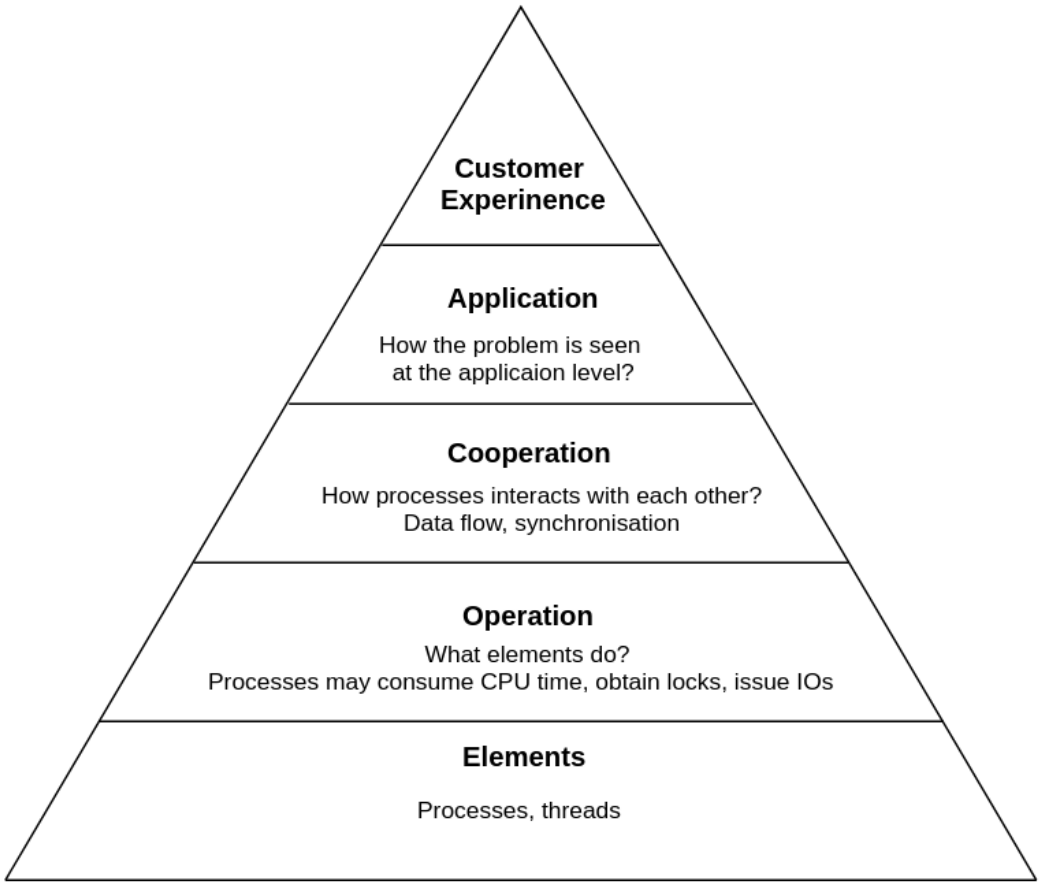
Sergey Kachkin


**Saint Petersburg**
3-4 November 2017

# Typical performance problem

- Poor problem description (“everything is slow”)
- Production environment
- No remote access to the machine
- Problem can be constant or intermittent
- Slowness is not reproducible in test environment

# Methodology




YA DRO

# Methodology

1. Any methodology is good
2. Focus on the application

# Linux trace data sources

- Static tracing
  - Tracepoints
  - User Defined Static Traces (USDT)
- Dynamic tracing
  - Kprobes (kprobes,jprobes,kretprobes)
  - Uprobes
- PMU

# ftrace

- Pros
  - Built-in since 2.6.28
  - Static and dynamic tracing
  - Predefined tracers for some subsystems
- Cons
  - No timed based CPU profiling
  - No analytics → needs frontend
  - Raw ascii data may be difficult to review

# perf

- Initially build for PMU counters
- May use static and dynamic trace points
- Works via single CLI command ‘perf’
- Can be used for ftrace and eBPF

# perf

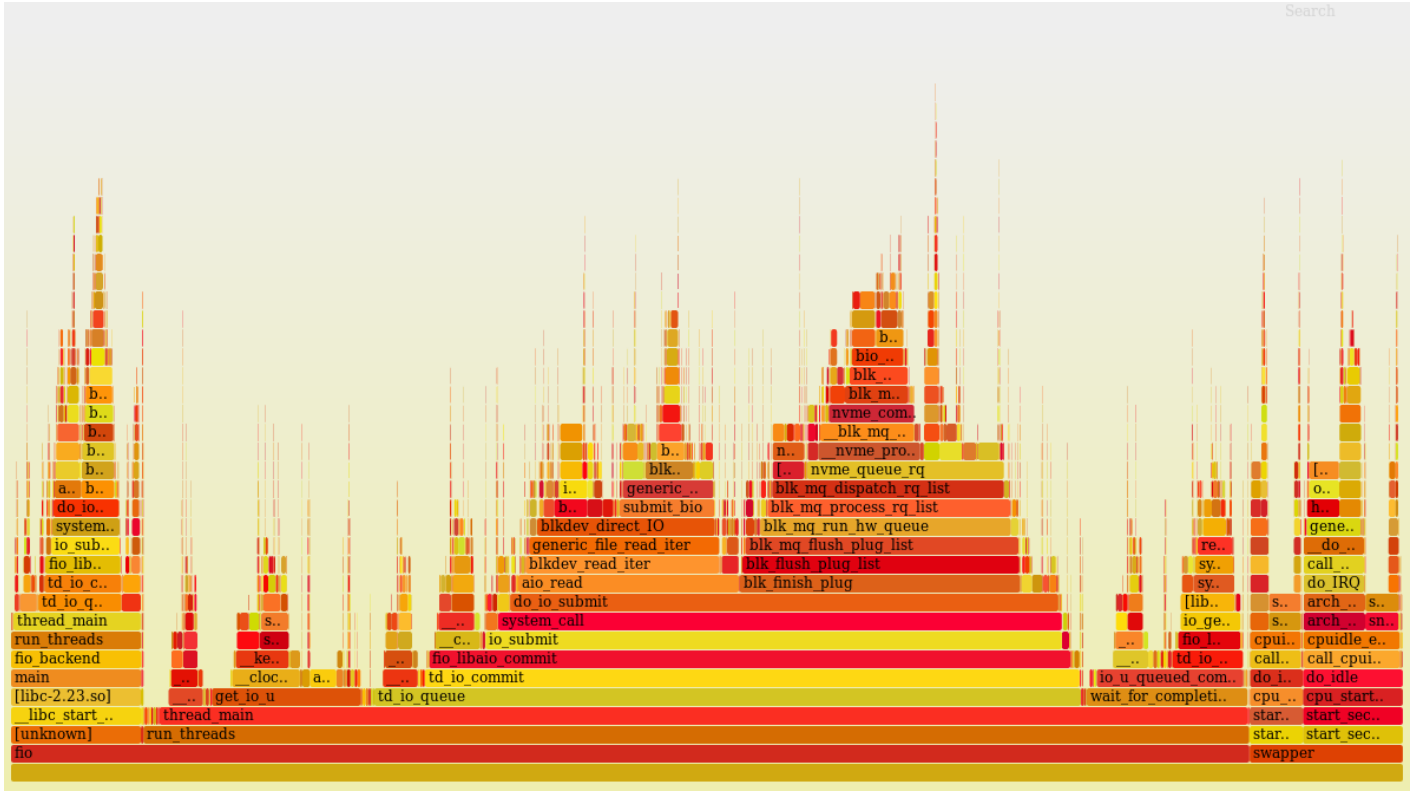
- Pros
  - Excellent for CPU profiling, system wide or PID
  - Backtraces with symbols: user, libraries, kernel, JIT
  - Nice opensource visualisations (flamegraph, heatmaps)
- Cons
  - Difficult to analyze processes cooperation
  - Limited analysis features

# perf real life example

- NVMe disk subsystem is much slower with LVM
- High system CPU utilization with LVM
- Collected perf CPU profile for ‘good’ and ‘bad’ cases

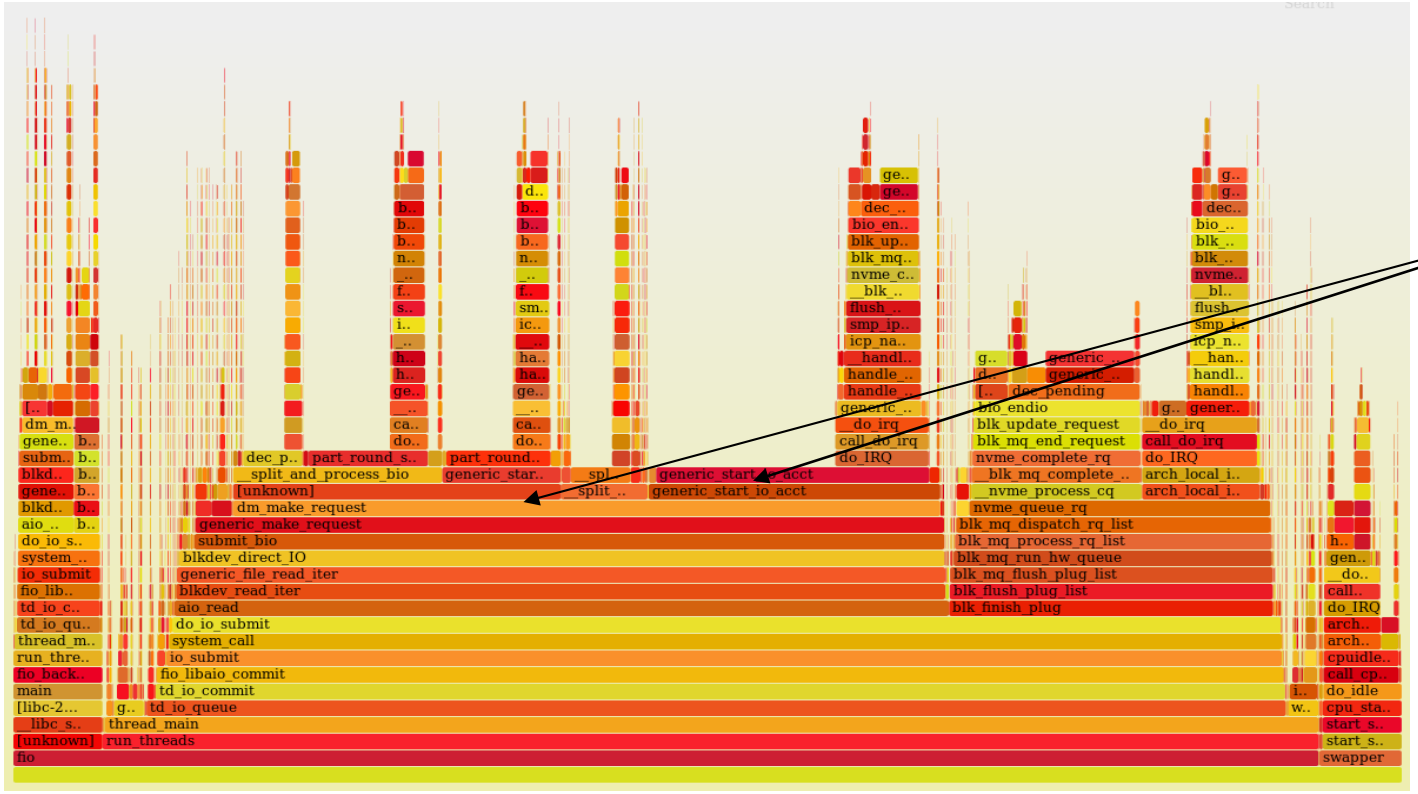
# perf real life example

- Fio benchmark against raw NVMe disks



# perf real life example

- Fio benchmark against NVMe with LVM




Functions to look at

# SystemTAP

- Pros
  - Very powerful for user and kernel space tracing
  - Many scripts available for common apps
  - In-kernel programming
  - Good for scientific research & complex debug
- Cons
  - Need high expertise to start
  - May be risky for production environment

# LTTng

- Pros
  - Good for monitoring production environments
  - Offline and online analysis options
  - Several tools for offline visualizations
  - Flight recorder mode
- Cons
  - No time based CPU sampling
  - Third party application with dependencies
  - Analytics with slow Python based scripts

# eBPF + BCC

- Pros
  - Many opensource scripts available
  - Python and Lua frontends in BCC
  - Safe for production as opposed to stap?
- Cons
  - Needs relatively new kernel 4.x
  - Needs coding to do simple tasks
  - May lose traces under high load

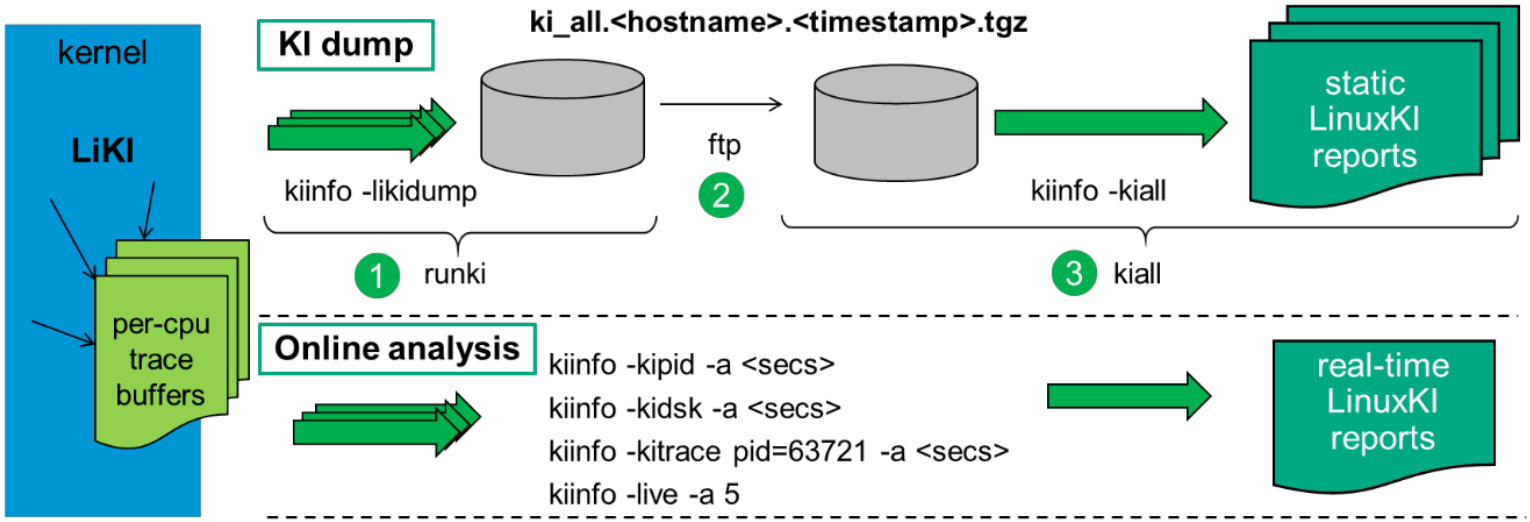
YA DRO

# LinuxKI

- Prerequisites
  - Kernels 2.6.32 through 4.12.8
  - Provided as .deb and .rpm package
  - Kernel headers may be needed to compile
  - Need root access

# LinuxKI design

- LIKI kernel module or ftrace as data source
- Online or offline reports with kiinfo tool




https://github.com/HewlettPackard/LinuxKI

# Performance mantra

- If it’s running, what’s it doing?
- If it’s waiting, what’s it waiting on?

# LinuxKI real life example

- Oracle is reporting high IO wait times
- Disk subsystem latency is normal
- High system CPU usage
- Collected LinuxKI dump for 20 seconds

# LinuxKI real life example

- Generated static report with ‘kiinfo -kipid’

```
PID 4042  ora_dbwb_mydb
  PPID 1  /usr/lib/systemd/systemd

    ********* SCHEDULER ACTIVITY REPORT ********
    RunTime     :  0.069893  SysTime    :  0.005610   UserTime    :  0.064283
    StealTime   :  0.000000
    SleepTime   : 19.681130  Sleep Cnt :         67   Wakeup Cnt :          1
    RunQTime    :  0.181381  Switch Cnt:         73   PreemptCnt :          6
    Last CPU    :         11  CPU Migrs :          5   NODE Migrs :          0
    Policy      : SCHED_NORMAL     vss :  25005274          rss :      22208

    busy     :       0.35%
      sys    :       0.03%
      user   :       0.32%
      steal  :       0.00%
    runQ     :       0.91%
    sleep    :      98.74%

    ******** PID RUNQ LATENCY REPORT ********
    RunQTime    :  0.181381  RunQCnt    :         73   AvRunQTime :  0.002485
    RunQPri     :  0.179702  RunQPriCt :         35   AvRunQPri  :  0.005134
    RunQIdle    :  0.001679  RunQIdleCt:         38   AvRunQIdle :  0.000044
```

# LinuxKI real life example

- Sleep report (off-cpu time)
- What the process is waiting for?

```
******** SLEEP REPORT ********

Kernel Functions calling sleep() - Top 20 Functions
   Count     Pct    SlpTime    Slp% TotalTime%   Msec/Slp   MaxMsecs  Func
      66  98.51%    18.2707  92.83%     91.66%    276.828   1235.465  read_events

Process Sleep stack traces (sort by % of total wait time) - Top 20 stack traces
   count    wpct      avg Stack trace
              %     msecs
================================================================
      58  79.32   269.171  read_events  sys_io_getevents  tracesys_phase2  unknown  |  0x0  skgfospo  skgfrwat
          ksfdwtio  ksfdwat_internal  ksfdwat  kcfwatwr  kcbbdrv  ksbabs  ksbrdp  opirip
       8  13.51   332.345  read_events  sys_io_getevents  system_call_fastpath  unknown  |  0x0
```

# LinuxKI real life example

- Cooperating threads report

```
******** COOPERATING/COMPETING TASKS REPORT ********

Tasks woken up by this task (Top 10)
       PID    Count   SlpPcnt     Slptime  Command
        53        1     5.44%    1.080792  [rcuos/6]

Tasks that have woken up this task(Top 10)
       PID    Count   SlpPcnt     Slptime  Command
     19322        1     6.28%    1.235465  [kworker/38:127]
     22049        1     4.67%    0.918277  [kworker/33:199]
     22255        1     3.96%    0.778919  [kworker/33:203]
     11314        1     3.28%    0.646519  [kworker/38:2]
     14346        1     3.23%    0.636643  [kworker/9:177]
     16518        1     3.19%    0.627862  [kworker/33:47]
     14186        1     3.09%    0.608355  [kworker/9:70]
     16949        1     2.63%    0.517456  [kworker/33:69]
     22012        1     2.63%    0.517263  [kworker/33:162]
     21988        1     2.62%    0.515373  [kworker/33:138]
```

# LinuxKI real life example

- CPU report (on-cpu profile)
- What the process is doing?

```
******** CPU ACTIVITY REPORT ********
The percentages below reflect the percentage
of the Thread's total RunTime spent in either
User code or System code
RunTime:   0.0699

  Count     USER      SYS     INTR
      6        5        1        0
           83.33%   16.67%   0.00%

HARDCLOCK entries
   Count      Pct  State  Function
       2   33.33%  USER   __intel_new_memcpy [/opt/oracle/product/12.1.0.2/bin/oracle]
       2   33.33%  USER   skgfrciohdlr [/opt/oracle/product/12.1.0.2/bin/oracle]
       1   16.67%  USER   kcbhpbwt [/opt/oracle/product/12.1.0.2/bin/oracle]
       1   16.67%  SYS    UNKNOWN

   Count      Pct  HARDCLOCK Stack trace
   ============================================================
       2   33.33%  skgfrciohdlr  skgfrliopo  skgfospo  skgfrwat  ksfdwtio  ksfdwat_internal  ksfdwat
                     kcfwatwr  kcbbdrv  ksbabs  ksbrdp  opirip  opidrv  sou2o  opimai_real  ssthrdmain
       2   33.33%  __intel_new_memcpy  kcfisio  kcbbdrv  ksbabs  ksbrdp  opirip  opidrv  sou2o
                     opimai_real  ssthrdmain  main  __libc_start_main
       1   16.67%  kcbhpbwt  kcbzpbuf  kcbb_prepare  kcbb_prepare_coalesce  kcbb_coalesce  kcbbwrcv
                     kcbbdrv  ksbabs  ksbrdp  opirip  opidrv  sou2o  opimai_real  ssthrdmain  main  __libc_start_main
       1   16.67%  unknown  |  __intel_new_memcpy  kcfisio  kcbbdrv  ksbabs  ksbrdp
                     opirip  opidrv  sou2o  opimai_real  ssthrdmain  main  __libc_start_main
```

# LinuxKI real life example

- Syscalls report

```
******** SYSTEM CALL REPORT ********
System Call Name      Count      Rate   ElpTime       Avg        Max    Errs    AvSz     KB/s
io_getevents              1       0.0 13.950218 13.950218  13.950218       0
   SLEEP                 59       2.9  15.611897   0.264608
      Sleep Func         58            15.611897   0.269171   0.918277  read_events
   RUNQ                                  0.042502
   CPU                                   0.001823
io_submit                 2       0.1   0.027574   0.013787   0.027305       0
   RUNQ                                  0.024014
   CPU                                   0.003560
   AIO Writes             1       0.0               13.977556  13.977556            8192     14.0
```

# LinuxKI real life example

- Filesystem related syscalls profile

```
******** FILE ACTIVITY REPORT ********
FD: 301 REG      dev: 0x10300000 /u01/big/mydb_database/DB/mydb_bigdata_21.dbf
System Call Name     Count     Rate     ElpTime        Avg        Max    Errs    AvSz     KB/s
io_submit                1      0.0    0.000000   0.000000   0.000000       0
   AIO Writes            1      0.0              13.977556  13.977556            8192     14.0

FD: 8 0-ukn    dev: 0x0 /proc/3997/stat
System Call Name     Count     Rate     ElpTime        Avg        Max    Errs    AvSz     KB/s
open                     1      0.0    0.000021   0.000021   0.000021       0
read                     1      0.0    0.000014   0.000014   0.000014       0     337      0.0
close                    1      0.0    0.000002   0.000002   0.000002       0
```

Single 8kb write took 13.9 seconds → looks bad

# LinuxKI real life example

- Disk IO activity report

```
Totals:
  Physical Writes:
    Cnt   :      91   Total KB:    3824.0    ElpTime:   1.45763
    Rate  :     4.5   KB/sec  :     190.9    AvServ :   0.01602
    Errs:         0   AvgSz   :      42.0    AvWait :   0.00000
    Requeue :          0   MaxQlen :         1
```

Physical IO is not showing any problem

# LinuxKI real life example

So, why Oracle dbw is waiting for kworkers?

```
PID 19322  [kworker/38:127]
  PPID 2  [kthreadd]

    ********* SCHEDULER ACTIVITY REPORT ********
    busy    :      0.16%
      sys   :      0.16%
      user  :      0.00%
      steal :      0.00%
    runQ    :      8.34%
    sleep   :     91.49%

    ******** SLEEP REPORT ********

    Process Sleep stack traces (sort by % of total wait time) - Top 20 stack traces
       count    wpct      avg Stack trace
                  %     msecs
    ================================================================
         77   5.72    13.617   md_flush_request  raid0_make_request  md_make_request  generic_make_request
              submit_bio  submit_bio_wait blkdev_issue_flush  ext4_sync_file  vfs_fsync_range  dio_complete
              dio_aio_complete_work  process_one_work  worker_thread  kthread  ret_from_fork  unknown
```

Conclusion: kworker suffers from scheduling delays and makes many barrier writes. The problem is gone after disabling barrier writes.

# LinuxKI pros and cons

- Pros
  - Easy to collect dump and generate reports
  - Portable data
  - Very detailed analytic
  - In most of cases enough to isolate the problem
- Cons
  - Needs installing a kernel module (may work via ftrace also)
  - Root privileges are needed
  - Generates a lot of disk IO to save a dump

- Questions?